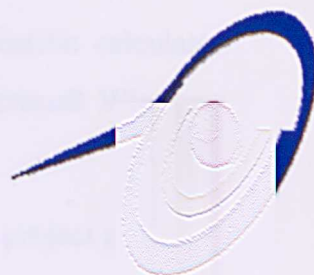




**Faculty of Computer Science and
Information Technology
University of Malaya**



e-Scientific Calculator

LIEW CHEE KIT

WEK 990011

**Under Supervision of
Assoc. Prof. Dr. Lee Sai Peck**

**Thesis Submitted by in Partial Fulfillment
of the Requirement for the
Degree of Bachelor of Computer Science
Session 2001/2002**

Abstract

This report is to specify the development of a software-based scientific calculator named eSciCalc. It is a standalone application running on the Microsoft Windows platform. There are nine chapters in this report.

The first chapter is the introduction to the project. Information of the project related to its background, motivation, problem definition, objective to achieve, project scope, expected outcome, and the proposed schedule is provided in this chapter.

The second chapter will present the result of literature review performed. The review on existing calculator and development tools are included here.

The third chapter will describe the development approach taken for this project. The strength of the approach is also included.

The fourth chapter states the result of requirements capture and analysis. The requirements are captured as use cases and the flow of even for each use case is included. In analysis, use case realisation in form of analysis classes is provided for each use case using collaboration diagrams and textual flow of event.

The fifth chapter will specify the software design. This chapter includes the use case realisation in term of design classes for each use case with a sequence diagram and textual flow of event.

Chapter six states the implementation approach employed in the project.

Chapter seven states the test model use in this project. This chapter include the test case and the test procedure that use to test the software.

Chapter eight states the evaluation of this project. This chapter include the strength, limitation and future enhancements of the software.

Chapter nine states the conclusion. Which include the problems and solutions, knowledge gained and project's conclusion.

Acknowledgement

I would like to express my deepest gratitude to my supervisor for this project, Associate Professor Dr. Lee Sai Peck, for spending her precious time on supervising me throughout the two semesters.

I would also like to express my special thanks to Mr Chiew Thiam Kian, my moderator for this project, for spending time to read my late report.

Table of Contents

Abstract i

Acknowledgement ii

Table of Contents..... iii

List of Figures ix

List of Tables..... xii

Chapter 1 Introduction 1

 1.1 Project Overview 1

 1.2 Motivations..... 2

 1.3 Objectives of the project 2

 1.4 Problem definition 3

 1.5 Project Scope..... 4

 1.6 Expected Outcome..... 4

 1.7 Project Schedule 5

 1.8 Chapter Summary..... 5

 1.9 Summary of Proposal..... 5

 1.10 References 6

Chapter 2 Literature Review..... 7

 2.1 Overview..... 7

 2.2 A Review of Existing Scientific Calculators..... 7

 2.2.1 CASIO S-V.P.A.M fx-570w Electronic Scientific Calculator..... 7

 2.2.2 Microsoft® Calculator version 5.0..... 12

 2.2.3 AllerCalc 2.11 13

 2.3 A Review of Development Tools..... 15

 2.3.1 Programming Languages..... 15

 2.3.1.1 Microsoft Visual Basic 6.0..... 15

 2.3.1.2 Java..... 15

 2.3.1.4 Microsoft Visual C++ 6.0..... 17

2.3.1.5	SQL	17
2.3.2	Componentisation.....	18
2.3.2.1	ActiveX	18
2.3.3	Relational Databases	20
2.3.3.1	MySQL.....	21
2.3.3.2	Microsoft Access 2000.....	22
2.3.3.3	SQL Server 2000.....	23
2.3.4	Database Connectivity.....	25
2.3.4.1	Open Database Connectivity (ODBC)	25
2.3.4.2	Java Database Connectivity (JDBC)	26
2.4	Proposed Tools.....	26
2.5	Chapter Summary.....	26
2.6	Reference.....	27
Chapter 3 Methodology		28
3.1	Software Development Approach	28
3.1.1	The Core Workflows.....	31
3.1.1.1	Requirements.....	31
3.1.1.1.1	Requirements Captures.....	31
3.1.1.1.2	Capturing Requirements as Use Cases.....	32
3.1.1.1.2.1	Activity: Find actors and use cases	33
3.1.1.1.2.2	Activity: Prioritise use cases	34
3.1.1.1.2.3	Activity: <i>Detail a use case</i>	35
3.1.1.1.2.4	Activity: Prototype user interface.....	36
3.1.1.1.2.5	Activity: Structure the use-case model.....	37
3.1.1.2	Analysis.....	38
3.1.1.2.1	Activity: Architectural analysis.....	39
3.1.1.2.1	Activity: <i>Analyse a use case</i>	40
3.1.1.2.1	Activity: <i>Analyse a class</i>	42
3.1.1.2.1	Activity: <i>Analyse a package</i>	43
3.1.1.3	Design.....	44
3.1.1.2.1	Activity: <i>Architectural design</i>	45
3.1.1.2.1	Activity: <i>Design a use case</i>	47

3.1.1.2.1 Activity: *Design a class* 48

3.1.1.2.1 Activity: *Design a subsystem* 50

3.1.1.4 Implementation..... 51

3.1.1.2.1 Activity: Architectural implementation..... 52

3.1.1.2.1 Activity: *Integrate System*..... 52

3.1.1.2.1 Activity: *Implement a subsystem*..... 53

3.1.1.2.1 Activity: *Implement a Class* 54

3.1.1.2.1 Activity: *Perform unit test*..... 55

3.1.1.5 Test..... 56

3.1.1.2.1 Activity: *Plan test*..... 57

3.1.1.2.1 Activity: *Design test*..... 57

3.1.1.2.1 Activity: *Implement test* 59

3.1.1.2.1 Activity: Perform integration test..... 60

3.1.1.2.1 Activity: Perform system test..... 61

3.1.1.2.1 Activity: Evaluate test 62

3.1.2 The Iteration Workflows 62

3.1.2.1 Inception iteration..... 62

3.1.2.2 Elaboration iteration..... 63

3.1.2.3 Construction iteration..... 63

3.1.2.4 Transition iteration 64

3.2 Strength of the proposed approach 64

3.3 Chapter Summary..... 66

3.4 References 66

Chapter 4 Requirements Capture and Analysis..... 67

4.1 Requirements Capture..... 67

4.1.1 Actor and Use Cases..... 67

Actor: User 67

Use Case: Perform General Calculation..... 67

Use Case: Perform Standard Deviation Calculation 69

Use Case: Saves Calculation to Memory List..... 70

Use Case: Retrieve Calculation from Memory List 71

Use Case: Retrieve value from the memory list..... 72

Use Case: Delete calculation from the memory list..... 73

Use Case: Retrieve Calculation from History List..... 73

Use Case: Retrieve Constant from Constant List..... 74

4.1.2 User Interface 74

4.1.3 Use-Case Model 77

4.2 Analysis..... 78

4.2.1 Use-Case Realisation – Analysis..... 78

Use Case: Perform General Calculation..... 78

Figure 4-8 and Figure 4-9 illustrates the use case realisation of use case Perform General Calculation..... 78

Use Case: Perform Standard Deviation Calculation 80

Use Case: Save Calculation to Memory List..... 80

Use Case: Retrieve Calculation from Memory List..... 81

Use Case: Retrieve Value from Memory List..... 82

Use Case: Delete Calculation from Memory List 82

Use Case: Retrieve Calculation from History List..... 83

Use Case: Retrieve Constant from Constant List..... 83

4.2.2 Analysis Class 84

4.3 Chapter Summary..... 87

4.4 References 87

Chapter 5 Design 88

5.1 Use-Case Realisation – Design..... 88

Use Case: Perform General Calculation..... 88

Use Case: Perform Standard Deviation Calculation 91

Use Case: Save Calculation to Memory List..... 92

Use Case: Retrieve Calculation from Memory List 94

Use Case: Retrieve Value from Memory List..... 95

Use Case: Delete Calculation from Memory List 96

Use Case: Retrieve Calculation from History List..... 97

Use Case: Retrieve Constant from Constant List..... 98

5.2 Design Class 99

5.3 Chapter Summary..... 101

5.4	References	101
Chapter 6 Implementation		102
6.1	Core components and development approach.....	102
6.1.1	Calculator screen	102
6.1.2	Expression classes	102
6.2	References	103
Chapter 7 Testing		104
7.1	Test Case	104
	Use Case: Perform General Calculation.....	104
	Use Case: Perform Standard Deviation Calculation	105
	Use Case: Save Calculation to Memory List.....	106
	Use Case: Retrieve Calculation from Memory List	106
	Use Case: Retrieve Value from Memory List.....	106
	Use Case: Delete Calculation from Memory List	107
	Use Case: Retrieve Calculation from History List.....	107
	Use Case: Retrieve Constant from Constant List.....	107
7.2	Test Procedure.....	108
	Test case supported: calculation statement $1 + (2+3) / \sin 20$	108
	Test case supported: standard deviation calculation statement 3, 4, 5.....	108
7.3	Chapter Summary.....	109
7.4	References	109
Chapter 8 Evaluation		110
8.1	Strength	110
8.2	Limitation.....	110
8.3	Future Enhancements	111
8.4	Chapter Summary.....	111
Chapter 9 Conclusion.....		112
9.1	Problems and Solutions.....	112
9.2	Knowledge Gained	112
9.3	Conclusion	113

Appendix A User Manual.....A-1

A.1 Overview of eSciCalc.....A-1

A.2 Features.....A-1

A. 3 Keyboard equivalents of calculator keys.....A-4

A.4 General calculation.....A-4

 A.4.1 Calculation.....A-5

 A.4.2 Answer display formatA-9

A.5 Standard deviation calculation.....A-9

A.6 Conclusion.....A-10

List of Figures

Figure 1-1 Fishbone diagram analysing the deficiency of a typical scientific calculator 2

Figure 1-2 Gant chart showing the schedule of this project..... 5

Figure 2-1 The relationships between components of Java 16

Figure 3-1 Phases further divided into more iterations. 29

Figure 3-2 Core workflows in iteration..... 29

Figure 3-3 Iterations in action 30

Figure 3-4 Emphasis shifts over the iterations, from requirements capture and analysis toward
design, implementation, and testing. 30

Figure 3-5 The workflow of capturing requirements as use cases 33

Figure 3-6 The input and result of activiyy finding actor and use cases..... 34

Figure 3-7 The input and result of activity prioritise use cases..... 35

Figure 3-8 The input and result of activity detail a use case 36

Figure 3-9 The input and result of activity prototype user interface..... 37

Figure 3-10 The input and result of activity structure the use-case model 38

Figure 3-11 The workflow in analysis 39

Figure 3-12 The input and result of activity architectural analysis..... 40

Figure 3-13 The input and result of activity analyse a use case..... 42

Figure 3-14 The input and result of activity analyse a class 43

Figure 3-15 The input and result of activity analyse a package..... 43

Figure 3-16 The activities in design workflow 45

Figure 3-17 The input and result of activity architectural design 47

Figure 3-18 The input and result of activity design a use case 48

Figure 3-19 The input and result of activity design a class..... 50

Figure 3-20 The input and result of activity design a subsystem..... 51

Figure 3-21 The activities in the implementation workflow..... 51

Figure 3-22 The input and result of activity architectural implementation..... 52

Figure 3-23 The input and result of ativity integrate system 53

Figure 3-24 The input and result of activity implement a subsystem 54

Figure 3-25 The input and result of activity implement a class 55

Figure 3-26 The input and result of activity perform unit test..... 56

Figure 3-27 The activities in workflow test 56

Figure 3-28 The input and result of activity plan test 57

Figure 3-29 The input and result of activity design test..... 59

Figure 3-30 The input and result of activity implement test 60

Figure 3-31 The input and result of activity perform integration test..... 61

Figure 3-32 The input and result of activity perform system test 61

Figure 3-33 The input and result of activity evaluate test..... 62

Figure 3-34 Models of the Unified Process..... 65

Figure 4-1 Statechart diagram showing the states of use case Perform General Calculation.. 69

Figure 4-2 Statechart diagram of use case Perform Standard Deviation Calulation..... 70

Figure 4-3 Statechart diagram showing the states of use case Saves Calculation to Memory
List..... 71

Figure 4-4 User interface design for use case Perform General Calculation 75

Figure 4-5 User interface design for use case Perform Standard Deviation Calculation..... 76

Figure 4-6 User interface design for the four lists..... 76

Figure 4-7 Use cse model of eSciCalc 77

Figure 4-8 Collaboration diagram describing key-in part of use case Perform General
Calculation..... 78

Figure 4-9 Collaboration diagram describing the execute expression part of the use case
model..... 79

Figure 4-10 Collaboration diagram describing use case Perform Standard Deviation
Calculation..... 80

Figure 4-11 Collaboration diagram describing use case Saves Calculation to Memory List .. 80

Figure 4-12 Collaboration diagram describing use case Retrieve Calculation from Memory
List..... 81

Figure 4-13 Collaboration diagram describing use case Retrieve Value from Memory List .. 82

Figure 4-14 Collaboration diagram describing use case Delete Calculation from Memory List
..... 82

Figure 4-15 Collaboration diagram describing use case Retrieve Calculation from History List
..... 83

Figure 4-16 Collaboration diagram describing use case Retrieve Constant from Constant List
..... 83

Figure 4-17 Class diagram showing the analysis classes of eSciCalc 86

Figure 4-18 The generalisation relations among the analysis classes	87
Figure 5-1 Sequence diagram describing use case Perform General Calculation.....	90
Figure 5-2 Sequence diagram describing use case Perform Standard Deviation Calculation .	91
Figure 5-3 Sequence diagram describing use case Save Calculation to Memory List.....	93
Figure 5-4 Sequence diagram describing use case Retrieve Calculation from Memory List ..	94
Figure 5-5 Sequence diagram describing use case Retrieve Value from Memory List.....	95
Figure 5-6 Sequence diagram describing use case Delete Calculation from Memory List	96
Figure 5-7 Sequence diagram describing use case Retrieve Calculation from History List....	97
Figure 5-8 Sequence diagram describing use case Retrieve Constant from Constant List.....	98
Figure 5-9 Class diagram showing the design classes	100
Figure 5-10 Class diagram showing the generalisation relationship of design classes.....	101
Figure A-1 eSciCalc calculator screen with standard notation and dual line display	A-2
Figure A-2 Select Hide keypad from the menu to hide the keypad.....	A-3
Figure A-3 General calculation window of eSciCalc.....	A-5
Figure A-4 Standard deviation calculation window of eSciCalc.....	A-10

List of Tables

Table 3-1 The set of activities for the requirement capture and their equivalent output 32

Table 3-2 Comparison of the use-case model and the analysis model..... 39

Table 3-3 Comparison of the analysis model and the design model..... 45

Table 4-1 Analysis classes and their responsibilities and attributes 85

Table A-1 The keyboard equivalent of the keys of eSciCalc.....A-4

1.1 Project Overview

Chapter 1 Introduction

This chapter will form the introduction to the project. The chapter will start with an overview of the project in Section 1.1. Then, the motivation will be presented in Section 1.2 while the objectives of the project will be presented in Section 1.3. The definition of the problem domain is stated in Section 1.4 “Problem Definition”, while the scope of the project is stated in Section 1.5. After defining the project scope, the expected outcome of this project is presented in Section 1.6 “Expected Outcome”, and the schedule of the project is stated in Section 1.7. The summary of this chapter and a summary of the following chapters are provided in Section 1.8 and Section 1.9 respectively. Finally, Section 1.10 will state the references for this chapter.

1.1 Project Overview

The outcome of this project is a standalone software-based scientific calculator named eSciCalc, which stands for eSciCalc.

This software-based calculator is implemented standalone rather than web based because we see scientific calculators as personal assistant. A scientific calculator should be highly available, provide fast response, and the data stored in the memory space should be personal and could be preserved for as long as the user needs it. A web-based application will only be available if the computer has access to the World Wide Web and when the server that hosts the application is up. As for last two reasons, which are to provide fast response and personal memory space, it would be difficult to achieve with a web-based application.

The mechanical calculating machine that is capable of performing addition, subtraction, multiplication and division operations has been introduced since the 1600s. However, the first hand-held scientific calculator was only introduced by Hewlett-Packard in 1972 [1].

After evolving for about thirty years, the scientific calculator available currently provides a very wide range of functions. Hence, adding more functions to the already wide collection is not the purpose of this project.

1.2 Motivations

This project is motivated by the discovery of a certain weaknesses of some existing scientific calculators. The main weakness identified is with the user interface.

In order to obtain a clearer view, problem analysis has been performed to find the deficiencies of a typical scientific calculator. The result is illustrated in Figure 1-1. The fishbone diagram in Figure 1-1 has outlined six problems of a typical scientific calculator. These problems will be further discussed in Section 1.4 “Problem Definition”.

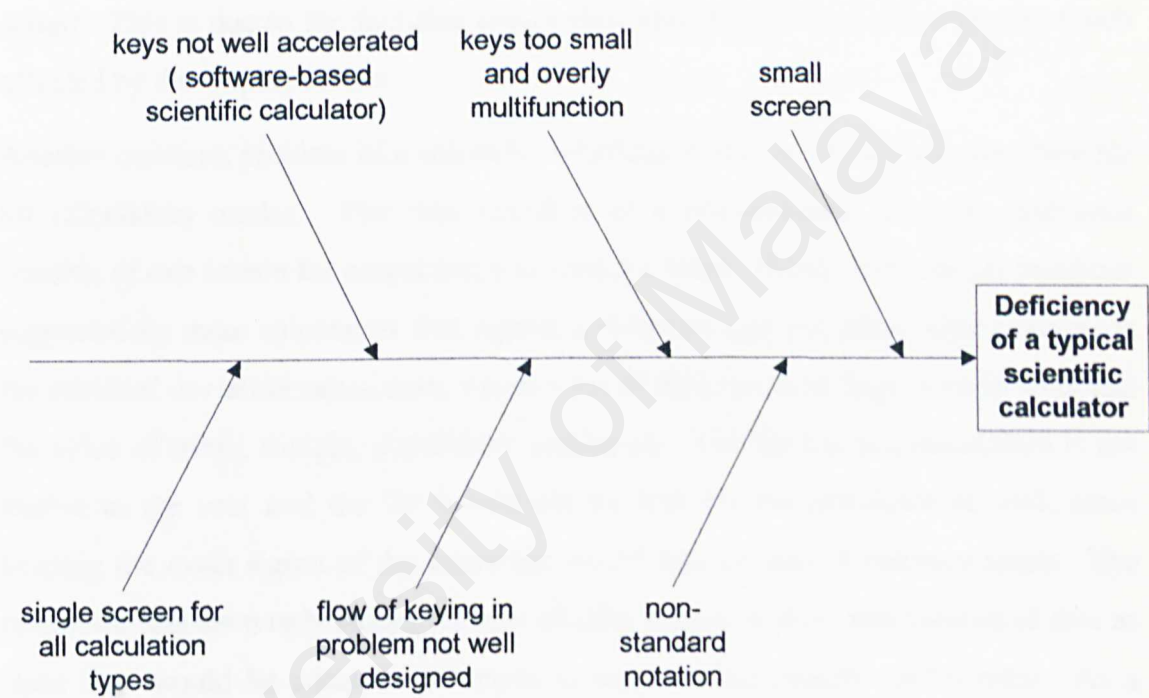


Figure 1-1 Fishbone diagram analysing the deficiency of a typical scientific calculator

1.3 Objectives of the project

The objective of this project is to:

- Provide a better user interface that would promote the use of standard notation, easy to redo, and reduce error rate and easy rechecking.
- Design with an easy to expand architecture. This would enable the product to support a more complete set of calculations.
- Provide a product that could reduce the use of papers and pen in calculations.

1.4 Problem definition

Out of the five problems stated in Figure 1-1, the problem tagged small screen is the biggest contributor to the user interface problem. This is due to the fact that the screen plays a lot of important roles. It is primarily used to display the result of calculations and to display the figures as the user key in the calculation. Besides, it is also used to indicate the current mode of the calculator.

As a result of this small screen playing such important roles, other problems such as flow of keying in problem not well designed, and the use of non-standard notation has arisen. This is due to the fact that key-in flow and the use of notation is very much affected by the display screen.

Another common problem of a scientific calculator is the use of one user interface for all calculation modes. The user interface of a conventional scientific calculator consists of one screen for output and a keypad for input. However, there are functions supported by these calculators that require a different user interface. One example is the standard deviation calculation, where a list of data has to be kept in order to obtain the value of mean, sample, population, and so on. The list for this calculation is not visible to the user and the list would not be kept by the calculator as well, since keeping the exact figure of the entire list would take up lots of memory space. The calculator will keep only values for sum of data, sum of square, and number of data as these data would be adequate for them to calculate the results user wanted. As a result, the user will never be able to confirm that the data keyed-in is totally the same with the list of data the user wants to calculate.

The final two problems identified are the keys being too small and overly multifunction and the keys for software-based calculators are not well accelerated. The former is due to the wide collection of functions that have to be supported while preserving the size of the calculator. As for the latter problem, keys for software-based scientific calculator should be well accelerated to enable the user to work faster. The user should not be made to press the numerous keys on the keypad of the calculator with the mouse.

1.5 Project Scope

The product of this project will be developed using the object-oriented method. The main purpose of using the object-oriented method is to realise the objective of designing with an easy to expand architecture. Besides, the product, which is a scientific calculator, would be made up of many classes to support the wide variety of calculation types and these classes have a high possibility of reuse.

This calculator will support two calculation modes. The first mode is named general calculation. This mode employs an **expression-based calculation**¹ method. All supported calculations that have an expression form including scientific calculations will be performed here. The second mode is the standard deviation mode. As the name suggests, calculations concerned with standard deviation will be performed in this mode.

Besides the three calculation modes, this calculator will provide three types of list to assist calculations. They are memory list, history list, and constant list. The memory list will provide fifty slots for calculations storage while the history list will automatically store the latest ten calculations.

1.6 Expected Outcome

The outcome of this project is expected to consist of the following.

- A standalone software-based scientific calculator running on the Microsoft Windows platform.
- An online help system and user's manual in compiled HTML (.chm) format.
- A printed user's manual.¹

¹ Expression-based calculation is a calculation method where the calculator does not calculate when the user is keying in the calculation. The calculator only performs calculation when the user presses the equal sign after the whole statement is keyed in. The calculator will first evaluate the expression for error. Error message will be shown if there is error in the expression, while the answer will be provided if there is no error.

1.7 Project Schedule

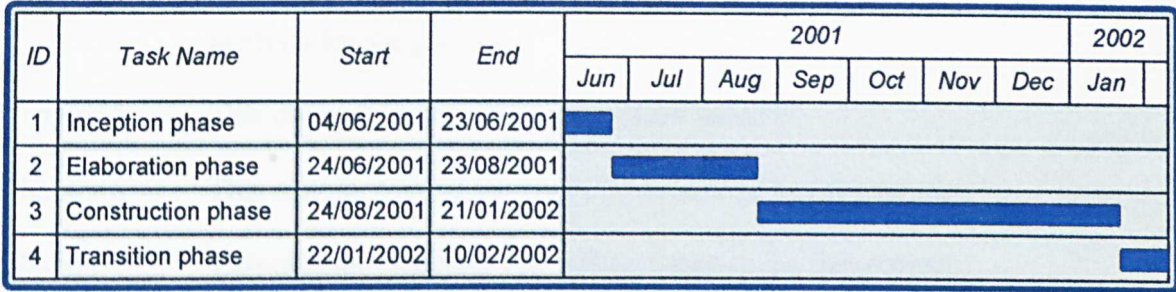


Figure 1-2 Gant chart showing the schedule of this project

Figure 1-2 illustrates the schedule of the project with a Gant chart. The tasks identified for this project follows the development approach in used. The approach is described in Section 3.1”Software Development Tools”.

1.8 Chapter Summary

The output of this project is a software-based standalone scientific calculator. The objectives that the project wants to achieve is to provide a better user interface, design with an easy to expand architecture and to reduce the use of papers and pen in calculations. The six problems identified with a conventional scientific calculator are the small display screen, the use of non-standard notation, key-in flow not well design, the use of one user interface for all calculations, having small and overly multifunction keys, and the keys of software-based calculator being not well accelerated. The product of this project will support two calculation modes: general calculation and standard deviation calculation, and three lists: memory list, history list and constant list. Besides the scientific calculator, a help system and users manual will be provided as the outcome of this project.

1.9 Summary of Proposal

- Chapter 1
- This chapter introduces the project.
- Chapter 2
- This chapter present the result of literature review performed on two areas: related product and options of development tools.

Chapter 3	This chapter provide a description on the methodology employed for this project.
Chapter 4	This chapter present the results of the requirement capture and analysis for the project.
Chapter 5	This chapter present the design of the project
Chapter 6	This chapter present the implementation part of the project.
Chapter 7	This chapter describes the software testing for this project.
Chapter 8	This chapter will form the evaluation of the software product.
Chapter 9	This chapter covers the conclusion of the project.
Appendix A	The user manual of eSciCalc.

1.10References

- [1] McGrath, K.A. (Ed). (1999).World of Invention. United States: Gale Research.

Chapter 2 Literature Review

This chapter documents the result of literature review for this project. The chapter starts with a brief overview in Section 2.1. After the overview, the reviews of existing calculators are described in Section 2.2. Section 2.3 will present the result of development tool reviews. The proposed tool is included in Section 2.4 after the review. Then, the summary of this chapter will be stated in Section 2.5, while Section 2.6 concludes the chapter with references.

2.1 Overview

Literature review is an analysis that is used to gather information about the system we intend to develop. This approach is used to evaluate existing system on the same topic so that a better product can be developed. It also includes the comparison of a few software, tools and approach to get the best outcome. Without this analysis, we would not be able to identify the strengths and weaknesses.

The literature review for this project will start with reviewing existing scientific calculators and followed by reviewing the various development tools in order to decide the best set of tools for this project.

2.2 A Review of Existing Scientific Calculators

Six scientific calculators have been reviewed but only three are specified in this section. This is because the other three do not impose any interesting features. One of the reviewed calculators is an electronic scientific calculator while the others are all software-based scientific calculator.

2.2.1 CASIO S-V.P.A.M fx-570w Electronic Scientific Calculator

The review of this calculator is carried out together with its user manual.

Modes

Basically there are five calculation modes (COMP, CMPLX, SD, REG and BASE), three angle unit modes (DEG, RAD and GRA), and three display modes (FIX, SCI and NORM with an additional ENG attribute).

The calculation modes are used to specify the main type of calculations to be performed. The five calculations modes and their description are as followed: -

- COMP: used to perform all the general calculations.
- CMPLX: the acronym refers to the word complex. This mode is used when performing complex number calculations.
- SD: the acronym stands for standard deviation. It is used to carry out standard deviation calculation.
- REG: the acronym refers regression. There are six types of regression provided in this calculator. The six types of regression calculation supported are Lin for linear, Log for logarithmic, Exp for exponential, Pwr for power, Inv for inverse, and Quad for quadratic.
- BASE: this mode is meant for calculation related to conversion between number systems and to perform logical operator calculations on them.

As for the angle units, all three are offered in this calculator. They are degree, which is labelled DEG, radiant labelled RAD and gradient being labelled GRA. Functions that perform calculation based on angles (e.g. trigonometry and hyperbolic) will produce different result with different angular modes being employed.

There are three modes offered for specifying the way answer is displayed. One of them is NORM, which reflects the word normal. It cancels the display effect that is provided when SCI or FIX is set and causes the answer display back to normal. The second mode is SCI, which refers to scientific. It enables the user to choose the number of significant digits to be displayed. Finally, it has FIX, which enables the user to set a fixed number of decimal points to be displayed. Only one of these three modes can be used at a time. Besides these three, there is one mode named ENG, which indicates the word engineering that can be used together with all the three previously revealed modes. It causes the answer displayed with terms such as Mega, kilo, micro, etc rather than the exponential way. In other words, 5000 will be displayed as 5 K rather than 5×10^3 .

Functionality

The general calculations that can be performed in the COMP modes are as follows:

- Basic calculations available on a standard calculator: addition (+), minus (−), multiplication (\times), division (\div), percentage (%) and negation (−)
- Memory: provides nine slots to store figures. They are labelled A, B, C, D, E, F, X, Y, and M. However, the addition and subtraction operations are offered for memory slot M only.
- Index: square (x^2), square root ($\sqrt{}$), cube (x^3), cube root ($\sqrt[3]{}$), power (x^y), root ($x^{\sqrt{}}$) and reciprocal (x^{-1}).
- Trigonometry and hyperbolic: sine (sin), cosines (cos), tangent (tan), arcsine (\sin^{-1}), arccosines (\cos^{-1}), arctangent (\tan^{-1}), and the button hyp to transform each trigonometric function into their equivalent hyperbolic function.
- Logarithms: common logarithm (log), common antilogarithm (10^x), natural logarithm (ln) and natural antilogarithm (e^x).
- Probability: factorial ($x!$), permutation (nP_r) and combination (nC_r).
- Integration: ($\int dx$).
- Coordinate: polar to rectangular and rectangular to polar.
- Constants: the more commonly used pie (π) shown as button. Forty more accessed through the button labelled CONST.
- Large number input: exponential (EXP), prefixex (Tera – T, Giga – G, Mega – M, kilo – k, mili – m, micro – μ , nano – n, pico – p, femto – f).
- Others: fraction (a^b/c and d/c), sexagesimal / decimal conversion ($^\circ ' ''$ and \leftarrow), random number generator (Ran#), angular unit change (DRG \blacktriangleright), forty units conversion (CONV)

As for the CMPLX mode which supports complex number calculations, most of the functions above are still offered since the real part as well as the figure part of the imaginary part can be made up of index, logarithms, trigonometric, hyperbolic, etc. functions. The functions specific to this mode are argument display (arg) and absolute display (Abs or $|z|$).

In the SD mode, the mode that supports standard deviation calculation, it provides data entry and data delete functions to enable the management of data in the list. The calculation of mean (\bar{x}), population ($x\sigma_n$) and sample ($x\sigma_{n-1}$) are automated too.

User Interface

Display screen: The display screen encompasses of two lines, the upper line is to enable the user to keep track of the calculation statement keying-in process while the bottom line is to display the answer to the calculation statement. The bottom part of the screen has indicators to specify the current modes and conditions the calculator is in.

The calculator works differently compared to the traditional calculator in the sense that it has a specific line for calculation statement input. The whole line of calculation statement is keyed in first and the calculator will execute the input to produce the answer when the equal sign is pressed. Implementing the calculation this way has reduced the error rate. This is due to the fact that the calculation statement will not be erased and is shown together with the answer. This will facilitate the user to make sure that the calculation statement is the one he/she wanted and the answer will be the correct one. This is different from the traditional method where the user will never know what had been keyed in.

Besides, having a different line for calculation statement key-in has also promoted easy undo and redo. Since the statement will not be erased after the answer is displayed, the user can easily edit the statement to the one he/she wanted. This has eliminated the need to re-key in the statement, which is almost the same when an error has been made as well as having to calculate a similar calculation with different set of figures.

In spite of all these advantages, the dual line display is still a single line input and single line output display. Hence, calculations that require a bigger display for input and output such as matrix are still very difficult to be implemented if not impossible. Even if they are being implemented, the notation would be rather confusing since it would have to be modified to suit this single line input and output. For example, the

fraction, which is being implemented in this calculator, the value for $2\frac{3}{4}$ is shown as $2\downarrow3\downarrow4$, while $2\downarrow3$ would be equivalent to $\frac{2}{3}$.

Moreover, the indicator at the bottom of the screen is a setback to its interface design. The indicator is small and is using acronym to a very extreme level. This is especially true with the angular measurement indicator, which is considered a vital information as the answer produced by the functions that uses angular units might not be the one user wanted if this mode is set wrongly. This calculator has uses the letter D for degree, R for radiant and G for gradient.

Besides, the standard deviation function support uses the same screen as all other functions. It faces the problem similar to the one stated in Section 1.4 “Problem Definition”. The user could not check the list to verify the answer.

Keypad: This model has promoted the grouping of key type. Key type grouping will accelerate key finding and will also endorse a user interface that is more understandable. This calculator has grouped the keys into three groups, one with the control keys such as arrows and shift, another with keys for scientific calculations, and the third group for arithmetic and statistical calculations.

Besides key grouping, the functions that require the press of the shift key for access are mostly the inverse of the original functions for that button. This designation of shift functions has also accelerated the finding of functions.

The designer has used colour to differentiate between functions that will be available in certain mode only. Colours have been a very good choice in this context since the area available is too small for any icon or similar things.

Notation: There are calculations in this calculator not following the standard mathematical notation. Among the contributor in to this problem is the implementation of functions that require a bigger display. Using non-standard notation would disrupt the work of its user besides being difficult to learn. One of the major objectives of user interface design is to enable users to fully concentrate on their work and not on how to use certain product to do their work. This non-standard notation will affect the user’s concentration and will increase error rate of one’s work as well.

Others: This calculator has a quick reference card as a companion. This card is placed in its cover to enable quick reference to functions that are impossible to be memorized. Among the information placed, here are the forty constants and their respected numbers for retrieval as well as the forty unit conversions with their retrieval numbers too. This is a very good way to compliment its inability to show the whole list for the user to choose from.

One of the major setbacks to interface design in this calculator is the extreme use of acronyms. This might be caused by the limited size for display, but the use of acronym has gone to an unacceptable level. Acronyms have been used in almost every part of the calculator. The names of the various types of modes are all in acronyms, the labels of buttons are using acronyms, and the error messages have acronyms as well.

2.2.2 Microsoft® Calculator version 5.0

Modes

There are three calculation modes in this calculator, the general calculation mode, standard deviation mode, and base-n calculation mode.

This calculator also supports the three angular measurement units, radians, gradients, and degrees.

Functionalities

As for functionalities, this calculator supports only the very basic ones. The scientific calculations supported are the trigonometric and hyperbolic, common and natural logarithms, and the indexes. In base-n, it provides the logical operations Mod (modulus), And (bitwise And), Or (bitwise Or), Xor (bitwise exclusive Or), Lsh (shift left), shift right and Not (bitwise inverse). The standard deviation mode provides only the average, sum, and standard deviation.

User Interface

Display screen: This calculator uses the single line displays. The mode indicators are not placed in the display screen. Instead, it uses radio buttons for mode change as well as mode indicator.

The standard deviation mode for this calculator has a separate window to store the list of data. Hence, the data of the list is visible to the user. However, data input and answers output still use the main screen of the calculator.

Keypad: There are not many keys on the keypad since the functions supported are very minimal. Keys grouping are utilised too but the grouping is rather confusing. One example is the keys for arithmetic calculation and the logical calculations being placed in one group. Another example is the placement of constant π together with the memory group.

Besides grouping, colours are used to further differentiate the keys too. However, there are labels of keys that have become hard to read due to the use of colour. This is especially true for the keys in scientific calculation group. The labels of those keys are light magenta in colour.

In spite of all the problems stated above, the keys of this calculator are all accelerated. This has enabled the user to perform calculations faster. However, mapping the functions of a scientific calculator keypad is a very subjective work. Different users might see the mapping of keys differently.

This calculator uses checkboxes in place of the hyperbolic and inverse keys use in most calculators. The tick in the checkbox will act as the indicator that the key has been pressed as opposed to the convention used by many calculators where the indicator is in the display screen.

2.2.3 AllerCalc 2.11

Modes

This calculator has only one mode, the scientific calculation mode. However, it supports all types of functions in one user interface and one key-in method.

Functions

The collection of functions supported by this calculator is quite large. It supports the scientific calculation, statistical calculation, and even financial calculation.

User Interface

Display screen: This calculator uses the expression-based calculation. It provides a big text area for calculation and all the calculations performed will stay in the text area until the user clears them. The content of the text area is named worksheet. The user could save the worksheet for future use. All the calculations performed could be saved as a file. However, due to the small font employed for this calculation text area, the screen would look very messy when the number of calculations appended increases.

Keypad: The keypad has a very minimum set of keys, while the other functions can be selected from the menu. As for keying-in, using the keyboard alone is enough since the expression is made up of characters and not a token. The user could just type in the functions one character at a time. Even though this may be a very flexible way, the user would have to memorise the functions and the number of parameters. If the wrong number of parameter is keyed in, the application will only tell you that the number of argument is wrong and does not tell the user how many arguments should there be. The application should at least give a brief description of the functions and provide the number of arguments as well as what those arguments stands for. Even in the help system, only a very small set of functions is described with the number of arguments. For most of the functions, only a one-line description of what the function returns is provided.

It follows Microsoft Calculators 5.0's way of dealing with inverse and hyperbolic functions, which uses the checkbox. As for the indicator for angular unit mode and number base mode, this calculator uses a place other than the screen for this purpose. Furthermore, the use of acronym is acceptable and the font size for these indicators is quite big. However, the user would again have to access the menu to change them.

Others: The menu system has been overly exploited in this calculator. The menu is used to change the angular unit, number base, answer display format, functions selection, and so on. Beside this, the arrangement is a bit messy as well. This is pointed to the menu "option". There are eighteen menu items in this menu alone and out of this eighteen, five menu items is a further drop down menu. The functions placed inside this menu include setting the display option, saving and loading of worksheets, setting the calculation options (angular unit, number base, number of

significant digit and display format), to show the unit conversion window and finance box, and to exit the application. Many of the functions in this menu should not be placed here. Examples are the save and load worksheet, the show unit conversion and finance box, and exit. They have nothing to do with options.

2.3 A Review of Development Tools

2.3.1 Programming Languages

2.3.1.1 Microsoft Visual Basic 6.0

Microsoft Visual Basic is a good tool for developing Windows applications with Graphic User Interface (GUI). There is a wide variety of components available to shape the user interface.

Microsoft Visual Basic is event-driven [1]; meaning code remains idle until called upon to respond to some event (button pressing, menu selection and so forth). An event processor governs Visual Basic. Nothing happens until an event is detected. Once an event is detected, the code corresponding to that event (event procedure) is executed. Program control is then returned to the event processor.

Microsoft Visual Basic 6.0 also supports the use of ActiveX components. In fact, it supports the development of ActiveX components as well. Besides, it has a set of powerful database access tools as well.

As for the case of this project, Microsoft Visual Basic 6.0 has insufficient support for scientific calculation.

2.3.1.2 Java

Java is a product of Sun Microsystems Inc. It is a programming language, a runtime system, a set of development tools and an application-programming interface (API) [2]. The relationships between these elements are depicted in Figure 2-1.

As illustrated in Figure 2-1, Java programs are written using predefined software packages of the Java API. The source code is compiled using the Java compiler into a form called compiled byte code, a form that can be executed on the Java virtual machine. The Java byte code is then interpreted as it is executed. The use of Java

virtual machine has enabled the compiled files to be executed on many platforms as long as the Java virtual machine for that platform is available and is installed on the computer. However, the Java virtual machine will take up some space. This is due to the additional software, such as dynamic link libraries, that are needed to implement the Java API on the operating system and hardware. For example, Java 2 Runtime Environment Standard Edition v1.3 will take up 20.8 MB for installation. Besides, the application developed will need a higher hardware requirement for runtime since it is not executed directly with the operating system and the computer hardware.

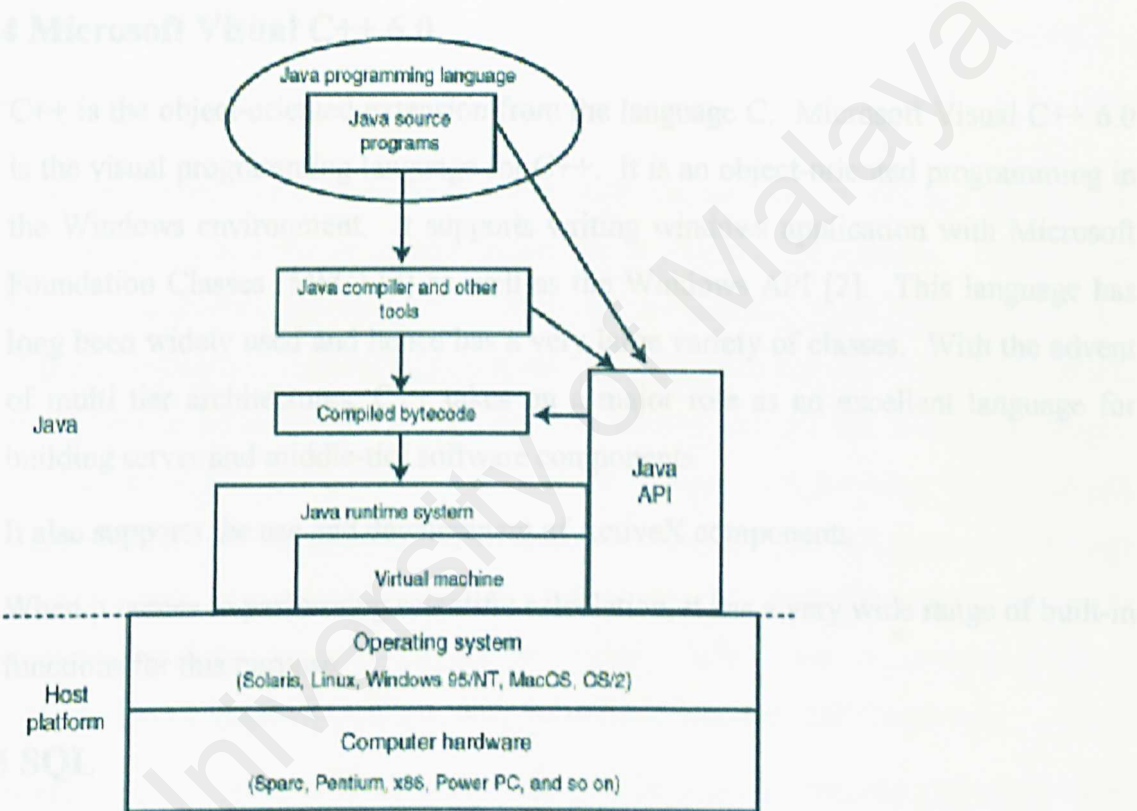


Figure 2-1 The relationships between components of Java

Java classes and objects directly support the object-oriented concepts of encapsulation, inheritance, messages and methods, and data hiding. Java interfaces provide support for multiple inheritance and polymorphism. The Java language retains all the benefits of object-oriented programming without the performance impacts associated with pure object languages, such as Smalltalk.

Java's object-oriented nature combined with numerous, compile-time and runtime integrity checks eliminate many difficult-to-find programming errors. The Java language has removed many of the dangerous programming capabilities, such as modifiable pointers, unchecked type conversion, and relaxed bounds checking.

The Java API provides extensive support of windowing and graphical user interface development without the complexities associated with maintaining multiple window class libraries. There are visual programming tools developed for Java.

As for this project, the class `java.math` is provided for mathematical calculations.

2.3.1.4 Microsoft Visual C++ 6.0

C++ is the object-oriented extension from the language C. Microsoft Visual C++ 6.0 is the visual programming language for C++. It is an object-oriented programming in the Windows environment. It supports writing windows application with Microsoft Foundation Classes (MFC) [3] as well as the Windows API [2]. This language has long been widely used and hence has a very large variety of classes. With the advent of multi tier architectures, C++ takes on a major role as an excellent language for building server and middle-tier software components.

It also supports the use and development of ActiveX components.

When it comes to performing scientific calculation, it has a very wide range of built-in functions for this purpose.

2.3.1.5 SQL

SQL is the de facto standard language used to manipulate and retrieve data from these relational databases. SQL enables a programmer or database administrator to do the following:

- Modify a database's structure
- Change system security settings
- Add user permissions on databases or tables
- Query a database for information
- Update the contents of a database

The most commonly used statement in SQL is the `SELECT` statement. Which retrieves data from the database and returns the data to the user. In addition to the `SELECT` statement, SQL provides statements for creating new databases, tables, fields, and indexes, as well as statements for inserting and deleting records.

Users will be able to program with SQL only on RDBMS databases that support SQL, such as MS-Access, Oracle, Sybase, and Informix. Although each vendor's implementation will differ slightly from the others, users should be able to use SQL with very few adjustments.

2.3.2 Componentisation

2.3.2.1 ActiveX

ActiveX referred to the conference slogan "Activate the Internet" and was more a call-to-arms than a technology or architecture for developing applications. ActiveX has become the all-encompassing term used to define everything from Web pages to OLE (Object Linking and Embedding) Controls. It has come to signify, on one hand, small, fast, reusable components that can get you hooked into all the latest technologies coming out of Microsoft, the Internet, and the industry. On the other hand, ActiveX represents Internet and applications integration strategies. ActiveX is not a technology or even architecture. It is a concept and a direction. The ActiveX components can be classified and broken into the six categories. Which are Automation Servers, Automation Controllers, Controls, COM objects, Documents and Containers.

Automation Servers are components that can be programmatically driven by other applications. An Automation Server contains at least one, and possibly more, *IDispatch*-based interfaces that other applications can create or connect to. An Automation Server may or may not contain User Interface (UI), depending on the nature and function of the Server. Automation Servers can be *in-process* (executing in the process space of the Controller), *local* (executing in its own process space), or *remote* (executing in a process space on another machine). The specific implementation of the server will, in some cases, define how and where the server will execute, but that is not guaranteed. A DLL can execute as in process, local or remote; an EXE can execute only locally or remotely.

Automation Controllers are those applications that can use and manipulate Automation Servers. A good example of an Automation Controller is VB. With the VB programming language, users are able to create, use, and destroy Automation Servers as though they are an integral part of the language. An Automation Controller can be any type of application, DLL or EXE, and can access the Automation Server either in-process, locally, or remotely. Typically, the registry entries and the implementation of the Automation Server indicate which process space the server will execute in relation to the Controller.

ActiveX Controls are equivalent to what is referred to as OLE Controls or OCXs. A typical Control consists of a UI representation both at design-time and runtime, a single *IDispatch* interface defining all of the methods and properties of the Control, and a single *IConnectionPoint* interface for the events that the Control can fire. In addition, the Control may have support for persistence across its execution lifetimes and support for various UI features, such as cut-and-paste and drag-and-drop features. Architecturally, a Control has a large number of COM interfaces that must be supported in order to take advantage of these features. With the release of the new OLE Control and ActiveX guidelines for Control development, a Control is no longer limited to the feature set defined in the preceding text. Rather, the developer can now choose to implement only those features that are most useful and interesting to users of the applications. ActiveX Controls always execute in process to the Container in which they reside. The extension of a Control is typically OCX, but in terms of execution models, it is nothing more than a standard window DLL.

COM Objects are similar in architecture to Automation Servers and Controllers. They contain one or more COM interfaces and probably little or no UI. These Objects, however, cannot be used by the typical Controller application the way Automation Servers can. The Controller must have specific knowledge of the COM interface that it "talks" to in order to use the interface, which is not the case for Automation interfaces. The Windows 95 and NT operating systems contain hundreds of COM Objects and Custom interfaces as extensions to the operating systems for controlling everything from the appearance of the desktop to the rendering of 3-D images on the screen. COM Objects are a good way to organize a related set of functions and data, while still maintaining the needed high-speed performance of a DLL.

ActiveX Documents, or DocObjects as they were originally called, represent Objects that are more than a simple Control or Automation Server. A document can be anything from a spread- sheet to a complete invoice in an accounting application. Documents, like Controls, have UI and are hosted by a Container application. Microsoft Word and Excel are examples of ActiveX Document Servers, and the Microsoft Office Binder and Microsoft Internet Explorer are examples of ActiveX Document Containers. The ActiveX Document architecture is an extension of the OLE Linking and Embedding model and allows the document more control over the container in which it is being hosted. The most obvious change is how the menus are presented. A standard OLE Document's menu will merge with the Container, providing a combined feature set; whereas an ActiveX Document will take over the entire menu system, thus presenting the feature set of only the document and not that of both the Document and the Container. The fact that the feature set of the Document is exposed is the premise for all the differences between ActiveX Documents and OLE Documents. The Container is just a hosting mechanism, and the Document has all of the control.

ActiveX Containers are applications that can host Automation Servers, Controls, and Documents. VB and the ActiveX Control Pad are examples of Containers that can host Automation Servers and Controls. The Microsoft Office Binder and the Microsoft Internet Explorer can host Automation Servers, Controls, and Documents. With the decreasing requirements defined by the ActiveX Control and Document specifications, a Container must be robust enough to handle the cases where a Control or Document lacks certain interfaces. Container applications may allow little or no interaction with the Document or Control they host, or they may provide significant interaction capabilities in both manipulation and presentation of the hosted component. This capability, however, is dependent upon the Container hosting the component and is not defined by any of the Container guidelines as being required.

2.3.3 Relational Databases

The concept behind the database is simple. A database is like a file cabinet that cans stores information. A database is a set of information related specific application.

Relational database management system and pronounced as separate letters, a type of database management system (DBMS) that stores data in the form of related tables. Relational databases are powerful because they require few assumptions about how data is related or how it will be extracted from the database. As a result, the same database can be viewed in many different ways.

An important feature of relational systems is that a single database can be spread across several tables. This differs from flat-file databases, in which each database is self-contained in a single table.

2.3.3.1 MySQL

MySQL, the most popular Open Source SQL database, is provided by MySQL AB. MySQL AB is a commercial company that builds its business providing services around the MySQL database.

MySQL is a database management system. A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL. Since computers are very good at handling large amounts of data, database management plays a central role in computing, as stand-alone utilities, or as parts of other applications.

MySQL is a relational database management system. A relational database stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility. The tables are linked by defined relations making it possible to combine data from several tables on request. The SQL part of MySQL stands for "Structured Query Language" - the most common standardized language used to access databases.

MySQL is Open Source Software. Open Source means that it is possible for anyone to use and modify. Anybody can download MySQL from the Internet and use it without paying anything. Anybody so inclined can study the source code and change it to fit their needs. MySQL uses the GPL (GNU General Public License) <http://www.gnu.org/>, to define what you may and may not do with the software in

different situations. If you feel uncomfortable with the GPL or need to embed MySQL into a commercial application you can buy a commercially licensed version from the company.

MySQL is very fast, reliable, and easy to use. MySQL was originally developed to handle very large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Though under constant development, MySQL today offers a rich and very useful set of functions. The connectivity, speed, and security make MySQL highly suited for accessing databases on the Internet.

2.3.3.2 Microsoft Access 2000

Access has existed in five main versions and one minor upgrade version. In the context of Access, a database can be viewed as a large repository in which table, report, form and other objects are stored.

The Microsoft Access package is one of the best selling relational database packages for Windows on the market. Microsoft has estimated that currently 10 millions people use this database package. Access provides two different modes. The first is an easy to use menu driven interface that let you issue commands without an in depth understanding of Access. Program mode lets the user to stored instruction in a Visual Basic file and executes them with one command.

Access allow user to indicate how tables should be related to each other. A table can have one-to-one, one-to-many or many-to-many relationship. A table that has referential integrity allows only one parent record for each child record. User can add, delete, and rearrange fields in the table structure. User can also control how the data will be entered in a table using the properties sheet of a field.

It is important to clarify the 'class' that Access fall into. Access is a desktop database package. It is not design to compete with system such as Oracle or SQL Server – full database servers – whose engines are superior in terms of speed and multi-user capabilities. This is usually the first perceived bad point. It does not provide a good performance when run across the network and more than a handful person using it at once. But it performance capability is good with limited multi-user capabilities. In

addition, it can and does make a good front-end package larger engine such as Oracle and SQL Server.

The other advantages are it is likely that you are running Windows as your operating system and using Microsoft Office as your application base. Access integrated well with these packages and data transfer between Access and the other Office components that are relatively easy.

2.3.3.3 SQL Server 2000

Business today demands a different kind of database solution. Performance, scalability, and reliability are essential, and time to market is critical. Beyond these core enterprise qualities, SQL Server 2000 provides agility to data management and analysis, allowing organization to adapt quickly and gracefully to derive competitive advantage in a fast-changing environment. From a data management and analysis perspective, it is critical to turn raw data into business intelligence and take full advantage of the opportunities presented by the Web. A complete database and data analysis package, SQL Server 2000 opens the door to the rapid development of a new generation of enterprise-class business applications that can give company a critical competitive advantage. The record-holder of important benchmark awards for scalability and speed, SQL Server 2000 is a fully Web-enabled database product, providing core support for Extensible Markup Language (XML) and the ability to query across the Internet and beyond the firewall.

SQL Server 2000 provides extensive database programming capabilities built on Web standards. Rich XML and Internet standard support give the ability to store and retrieve data in XML format easily with built-in stored procedures. User can also use XML update programs to insert, update and delete data easily.

- Easy access to data through the Web. With SQL Server 2000, you can use HTTP to send queries to the database, perform full-text search on documents stored in database, and run queries over the Web with natural language.
- Powerful, flexible Web-based analysis. SQL Server 2000 Analysis Services capabilities are extended to the Internet. User can access and manipulate cube data by means of a Web browser.

Achieve unparalleled scalability and reliability with SQL Server 2000. With scale up and scale out capabilities, SQL Server meets the needs of demanding commerce and enterprise applications.

- Scale up. SQL Server 2000 takes advantage of symmetrical multiprocessor (SMP) systems. SQL Server Enterprise Edition can use up to 32 processors and 64 GB of RAM.
- Scale out. Scale out distributes the database and data load across servers.
- Availability. SQL Server 2000 achieves maximum availability through enhanced fail over clustering, log shipping, and new backup strategies.

SQL Server 2000 is the data management and analysis backbone of the Microsoft .NET Enterprise Servers. SQL Server 2000 includes tools to speed development from concept to final delivery.

- Integrated and extensible analysis services. With SQL Server 2000, user can build end-to-end analysis solutions with integrated tools to create value from data. Additionally, user can automatically drive business processes based on analysis results and flexibly retrieve custom result sets from the most complex calculations.
- Quick development, debugging, and data transformation. SQL Server 2000 features the ability to interactively tune and debug queries, quickly move and transform data from any source, and define and use functions as if they were built in to Transact-SQL. Users can visually design and code database applications from any Visual Studio tool.
- Simplified management and tuning. With SQL Server 2000, it is easy to manage databases centrally alongside all enterprise resources. Stay online while easily moving and copying databases across computers or between instances.
- Ms SQL Server 7.0 is outperformed than MS Access and Informix SQL. This is because it includes a superset the ASNI standard SQL language elements that could not be find in MS Access and Informix [4].

2.3.4 Database Connectivity

2.3.4.1 Open Database Connectivity (ODBC)

ODBC allows a single uniform language to access different databases, instead of using the propriety language of each database by designing a standard set of APIs. Each database has its own API and it will interpret any request by the programmer so that the database can return the information. This open connectivity to a database allows an application to get data from any kind of database by using the appropriate ODBC driver.

A driver usually contains the callable API functions for a single database. The drivers are dynamic link libraries (DLLs) and the Driver Manager (ODBC DSN Administrator) is an executable program. A Data Source Name (DSN) must be created in order the driver could locate where is the database. In ODBC version 4.0 of the Administrator, there are three choices of DSN that can be created there are:

- System DSN allows every user of the computer and every system-level resource access to that database.
- File DSN allows all users to access to the same drivers.
- User DSN allows only the specific user to access the database.

And the ODBC drivers are available in ODBC version 4.0 are,

- Microsoft Access
- Microsoft dBase
- Microsoft Excel
- Microsoft FoxPro
- Microsoft ODBC for Oracle
- Microsoft Paradox
- Microsoft Text
- SQL server

2.3.4.2 Java Database Connectivity (JDBC)

JDBC (Java Database Connectivity) technology is an API (Application Program Interface) that let user access virtually any tabular data source from the Java programming language. It provides cross-DBMS connectivity to a wide range of SQL databases, and now, with the new JDBC API, it provides access to other tabular data sources, such as spreadsheets or flat files.

JDBC is modelled on ODBC (Object Database connectivity) but in addition provides an object-oriented model for accessing databases, permitting use of Java methods as well as SQL for querying and updating data. The JDBC standard means that applications can be written without considering what driver will be used in the final deployment, and gives system managers the freedom to change database engines without requiring a change in program logic.

The JDBC API allows developers to take advantage of the Java platform's "Write Once, Run Anywhere" capabilities for industrial strength, cross-platform applications that require access to enterprise data. With a JDBC technology-enabled driver, a developer can easily connect all corporate data even in a heterogeneous environment [5].

2.4 Proposed Tools

The tools that will be used for this project includes:

- Microsoft Visual Basic 6.0 as the core language especially used for the creation of user interface.
- Microsoft Visual C++ 6.0 for the development of ActiveX components. These ActiveX components will be use in Microsoft Visual Basic 6.0 to compute scientific calculation.
- Microsoft Access 2000 database for storing data.

2.5 Chapter Summary

This chapter has reviewed three scientific calculators and numerous software development tools. The review of scientific calculators has provided an input to the subsequent steps. As for software development tools, Microsoft Visual Basic 6.0 has

been chosen to produce the user interface while Microsoft Visual C++ will be used to develop ActiveX components. Microsoft Access 2000 has been chosen as the database for this project.

2.6 Reference

- [1] Balena, F.(1999).Programming Microsoft Visual Basic 6.0. Washington:Microsoft Press.
- [2] Petzold, C. (1998).Programming Windows,(5th ed.). Washington: Microsoft Press
- [3] Prosise, J. (1999).Programming Windows with MFC. (2nd ed.). Washington: Microsoft Press
- [4] Microsoft SQL Server - Product Overview. 22 Aug.2001 <<http://microsoft sql server - product overview.htm/>>.
- [5] JDBC Driver\JDBC(TM) Technology. 22 Aug.2001 <[http://jdbc driver/JDBC\(TM\) Technology.htm](http://jdbc driver/JDBC(TM) Technology.htm)>

Chapter 3 Methodology

This chapter provide a description of the development approach used for this project. The main section for this chapter is Section 3.1, which describes the approach. Section 3.2 will provide an outline of the strength of this approach. Then, Section 3.3 will summarise the chapter while Section 3.4 provides the references for this chapter.

3.1 Software Development Approach

The software development approach that has been employed for this project is named the Unified Software Development Process or in shorts the Unified Process. The Unified Process is component-based, which means that the software being built is made up of software components interconnected via well-defined interfaces. Besides, the Unified Process uses Unified Modelling Language (UML) when preparing all blueprints of the software.

The Unified Process is an iterative and incremental life cycle model. The cycle is made up of two distinct types of workflows. They are named the core workflows and the iteration workflows. There are five tasks classified as core workflows: requirements, analysis, design, implementation, and test. As for the iteration workflows, there are four phases: inception, elaboration, construction, and transition.

In the Unified Process, the development process is performed as iterations. The phases in the iteration workflows can be carried out in one iteration or divided into more iteration depending on the project. Figure 3-1 shows the whole cycle, which consists of four phases being divided into more iterations. Within every iteration, the five core workflows will be performed as shown in Figure 3-2. Hence, all the nine iterations in Figure 3-1 will sweep through the five core workflows sequentially as shown in Figure 3-3.

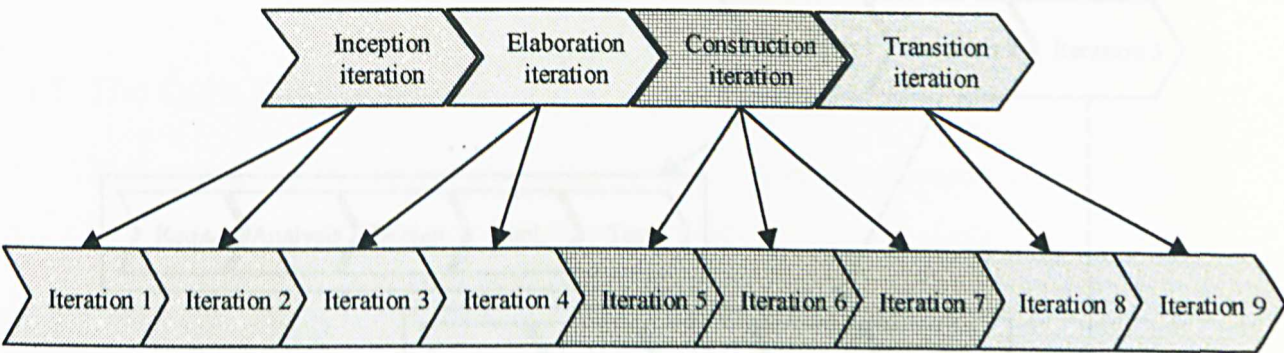


Figure 3-1 Phases further divided into more iterations.

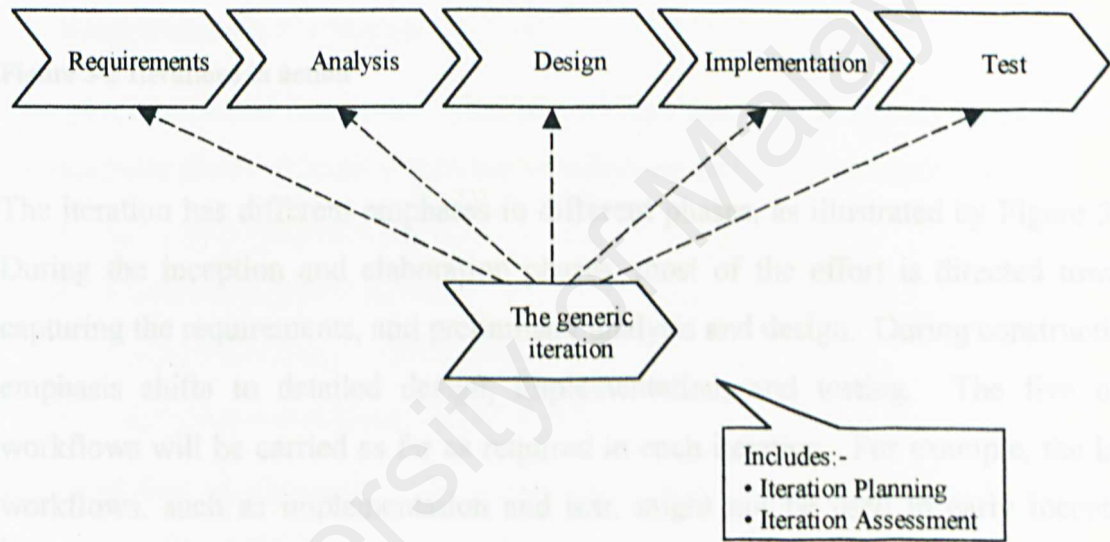


Figure 3-2 Core workflows in iteration

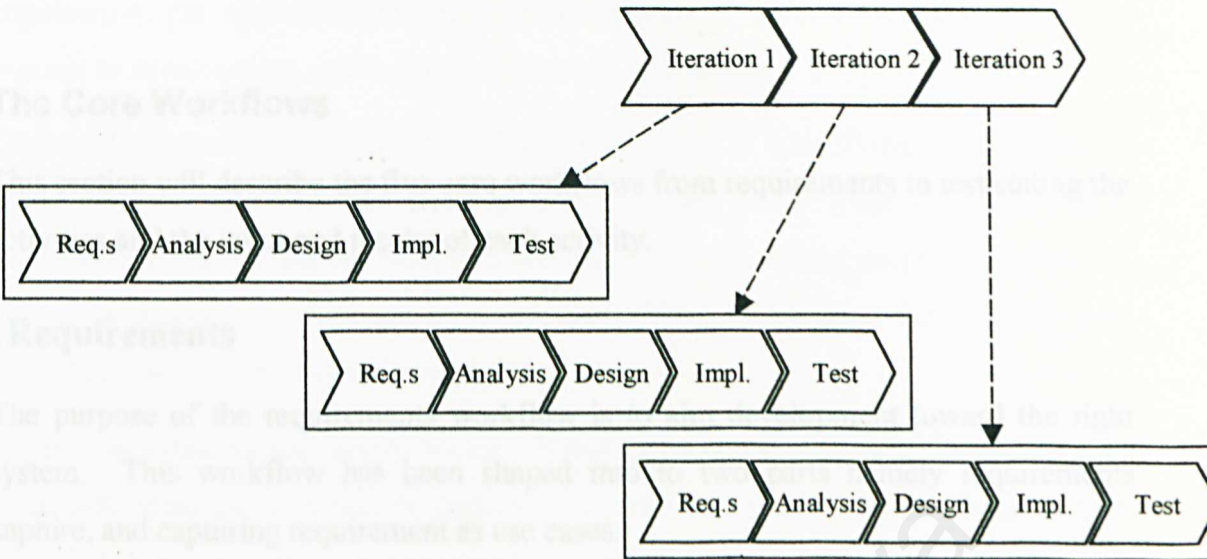


Figure 3-3 Iterations in action

The iteration has different emphases in different phases, as illustrated by Figure 3-4. During the inception and elaboration phases, most of the effort is directed toward capturing the requirements, and preliminary analysis and design. During construction, emphasis shifts to detailed design, implementation, and testing. The five core workflows will be carried as far as required in each iteration. For example, the later workflows, such as implementation and test, might not be used in early inception phase.

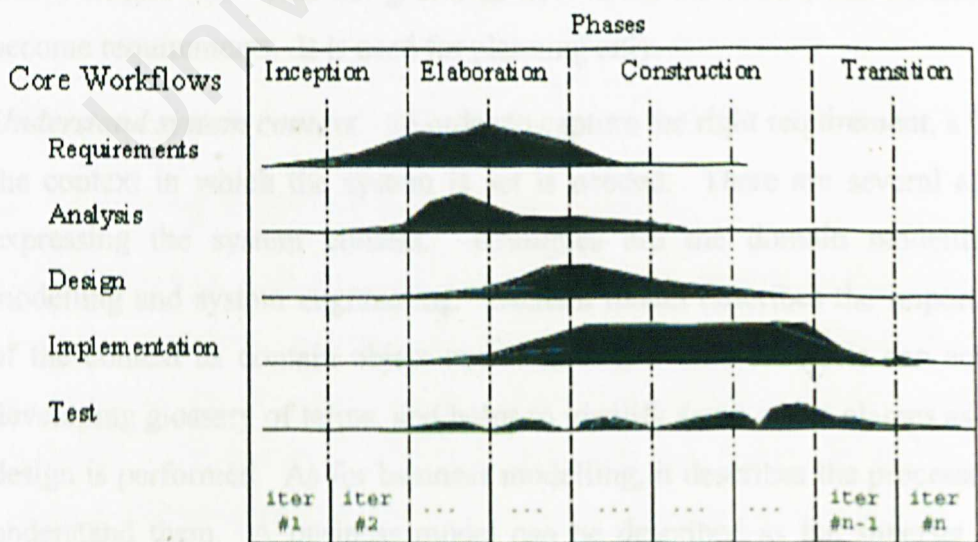


Figure 3-4 Emphasis shifts over the iterations, from requirements capture and analysis toward design, implementation, and testing.

3.1.1 The Core Workflows

This section will describe the five core workflows from requirements to test stating the activities and the input and results of each activity.

3.1.1.1 Requirements

The purpose of the requirements workflow is to aim development toward the right system. This workflow has been shaped into two parts namely requirements capture, and capturing requirement as use cases.

3.1.1.1.1 Requirements Captures

This part consists of four tasks. This part of the requirements is only emphasised in the inception phase. The four tasks are listed below and their description follows.

- List candidate requirements
- Understand system context
- Capture functional requirements
- Capture non-functional requirements

List candidate requirement This occurs when stakeholders come up with good ideas that might turn into requirements. These candidate requirements are kept in a list called feature list. This list grows as new items are added, and shrinks as features become requirements. It is used for planning only.

Understand system context In order to capture the right requirement, a firm grasp of the context in which the system is set is needed. There are several approaches to expressing the system context. Examples are the domain modelling, business modelling and system engineering. Domain model describes the important concepts of the context as domain object and their links. These objects can act as input to developing glossary of terms, and helps to identify some of the classes as analysis and design is performed. As for business modelling, it describes the processes in order to understand them. A business model can be described as the superset of a domain model because it also establishes the competency, which is crucial when identifying use cases, apart from identifying objects for the software system. As for system

engineering, the system is divided into subsystems.... However, its result is not useable in development, unlike the previous two approaches.

Capture functional requirements The primary way of identifying requirements is based on use cases. Each use case represents one way of using the software system. This step will be carried out with interviewing users, discussing proposal, and so on. Hence, in conjunction with acquiring use cases, the user interface for each use case should also be specified with the users.

Capture non-functional requirements Non-functional requirements that are specific to an individual use case will be capture in the use case model. As for those that are more generic and cannot be connected to a particular use case or a particular real-world class will be managed separately in a list of supplementary requirements.

Use cases can be used to capture functional requirements as well as non-functional requirements that are specific to their respective use cases.

The output of these four steps are summarize in Table 3-1.

Workflows	Resulting artifacts
List candidate requirements	Feature list
Understand system context	Business model or domain model
Capture functional requirements	Use-case model
Capture non-functional requirements	Supplementary requirements or Individual use cases (for use-case specific requirements)

Table 1 The set of activities for the requirement capture and their equivalent output

3.1.1.1.2 Capturing Requirements as Use Cases

The workflow capturing requirements as use cases consist of five activities: finding actors and use cases, prioritise use cases, detail a use case, prototype user interface, and structure the use-case model. These five activities and the logical flows are illustrated in Figure 3-5. The path in Figure 3-5 shows the logical sequence of activities using results from the previously performed activity as input.

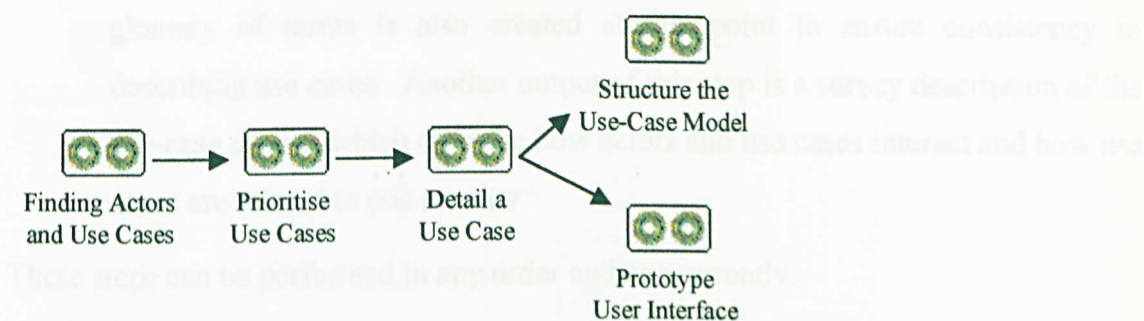


Figure 3-5 The workflow of capturing requirements as use cases

3.1.1.1.2.1 Activity: Find actors and use cases

This is the most essential activity for getting the requirements right. This activity has a business model or domain model, a supplementary requirements list, and a feature lists as input as illustrated in Figure 3-6. Besides, input from customer and users are needed too. The product of this activity is a use-case model that is described and diagrammed superficially to the extend where each use case can be described in detail, and a glossary of terms. There are four steps to be taken for this activity:

- **Finding the actors.** During this step, all types of user of the system and all external systems with which the system interacts with are identified. For each of the identified actor, a name, a brief description of its role and what it uses the system for is appended.
- **Finding the use cases.** The actors identified in the previous steps are used to identify candidate use cases for each actor. Besides, candidate use cases may also come from customers and users. The candidate use cases are then revised to produce a set of use cases that has an appropriate scope. The use cases often need to be restructured a few times before the use-case model stabilizes.
- **Briefly describing each use case.** During this step, each use case is briefly described. The description consists of a few sentences that summarise the actions, and a step-by-step description of what the system need to do when interacting with its actors.
- **Describing the use-case model as a whole.** For this step, diagrams and description to explain the use-case model as a whole is prepared with emphases on how the use cases relate to each other and to the actors. A

glossary of terms is also created at this point to ensure consistency in describing use cases. Another output of this step is a survey description of the use-case model, which describe how actors and use cases interact and how use cases are related to one another.

These steps can be performed in any order and concurrently.

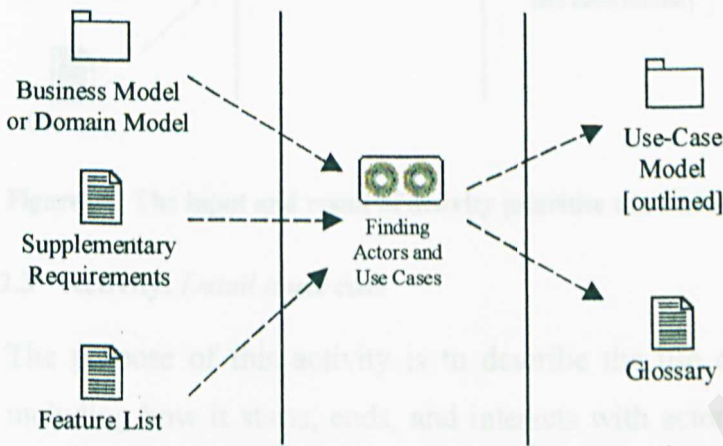


Figure 3-6 The input and result of activity finding actor and use cases

3.1.1.1.2.2 Activity: Prioritise use cases

The purpose of this activity is to determine which use cases need to be developed in early iterations, and which can be developed in later iterations. As illustrated in Figure 3-7, this activity has the supplementary requirements, the use-case model [outlined] and the glossary as input. The results of this activity are captured in an architectural view of the use case model. This view has to depict the architecturally significant use cases.

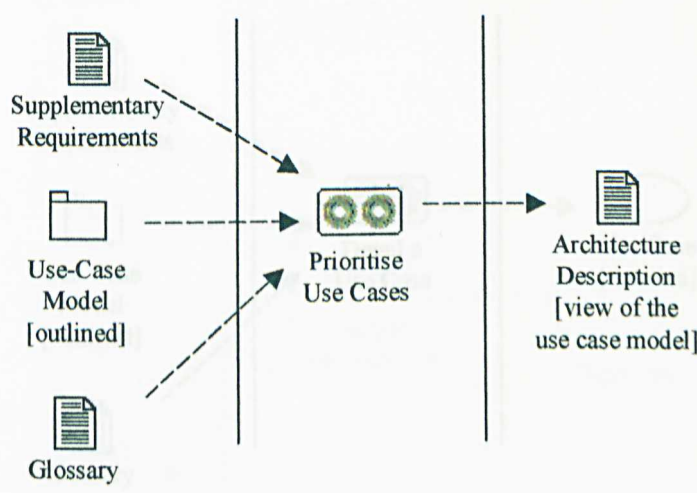


Figure 3-7 The input and result of activity prioritise use cases

3.1.1.1.2.3 Activity: *Detail a use case*

The purpose of this activity is to describe the use case’s flow of events in detail, including how it starts, ends, and interacts with actors. This step is performed with users of the use cases. These users will be the source of description and to review the use-case description to verify them. As illustrated by Figure 3-8, this activity will have the supplementary requirements, use-case model [outlined], and a glossary as a starting point. The result of this activity is a detailed description of a particular use case in text and diagrams. This activity ...

- **Structuring the use-case description.** In this step, the states that the use-case instances and the possible transition between those states are described. Out of the many possible transitions, one complete basic path is first described. The rest of the paths will be described in a different section as alternatives or deviations from the basic path. If however that a particular alternative or deviation path is small enough to be described in one line, it can be included in the basic path description. Besides these paths, the pre-condition and post-condition of the use case should be defined as well.
- **Formalising the use-case description.** For use cases that have a very large number of states and alternative transitions it would become too complex to describe consistently with text, a more structured description technique can be used. There are three types of diagrams in UML that can be used for this purpose: statechart diagram, activity diagram, and interaction diagram.

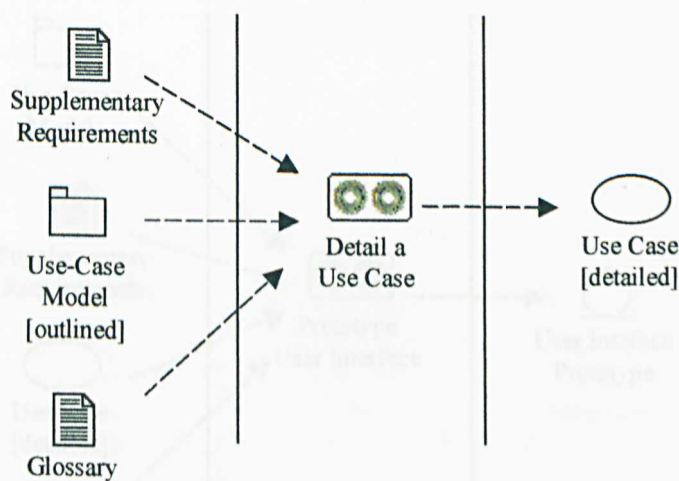


Figure 3-8 The input and result of activity detail a use case

3.1.1.1.2.4 Activity: Prototype user interface

The purpose of this activity is to build the user interface prototype. This activity has the use-case model, supplementary requirements, use case description, and the glossary as the starting point as illustrated in Figure 3-9. This activity will produce a set of user interface sketches and prototypes for the most important actors. This activity is carried out in two steps stated as followed.

- **Creating logical user interface design.** This step identified the user interface elements that are needed for users to interact with a use case. The use cases will be going through one by one to identify the proper user interface elements for each use case.
- **Creating physical user interface design and prototype.** During this step, sketches of user interface elements combined to form the physical user interface. Then, executable prototypes are built for the important user interface elements. These sketches and prototypes will be validated through user interface review and will work as a specification of the user interface when the real user interface is being constructed.

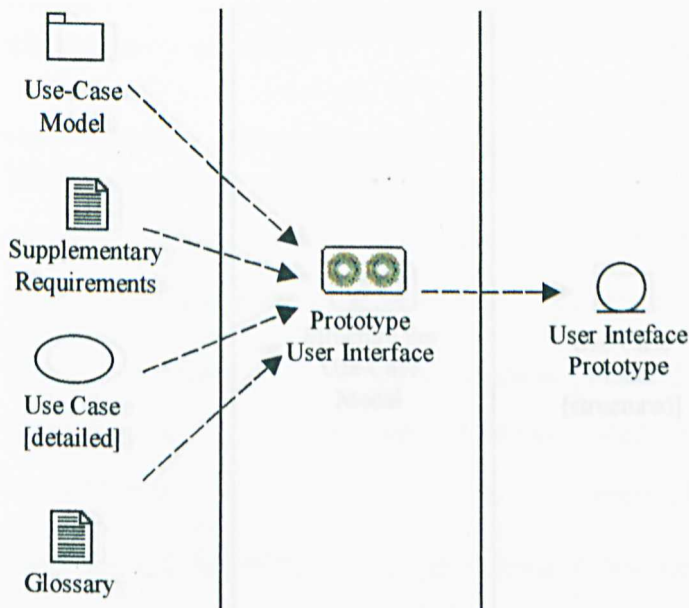


Figure 3-9 The input and result of activity prototype user interface

3.1.1.1.2.5 Activity: Structure the use-case model

This activity is taken to extract general and shared use-case description of functionality that can be used by more specific use-case descriptions, and to extract additional or optional use-case description of functionality that can extend more specific use-case description. As illustrated in Figure 3-10, the use-case model, the supplementary requirements, the detailed use cases, and the glossary will be used to accomplish this activity. There are three steps to be taken in this activity.

- **Identifying shared description and functionality.** The actions or part of actions that are common to or shared by several use cases will be identified. This sharing is then extracted and described in a separate use case that can then be reused by the original use cases through the uses relationship (uses relationship is the name given for generalisation relationship in use case model).
- **Identifying additional and optional description of functionality.** In this step, additional and optional description of functionality are identified to model them using the extend relationship.
- **Identifying other relationships between use cases.** In this step, the include relationship is used to further structure the use-case model.

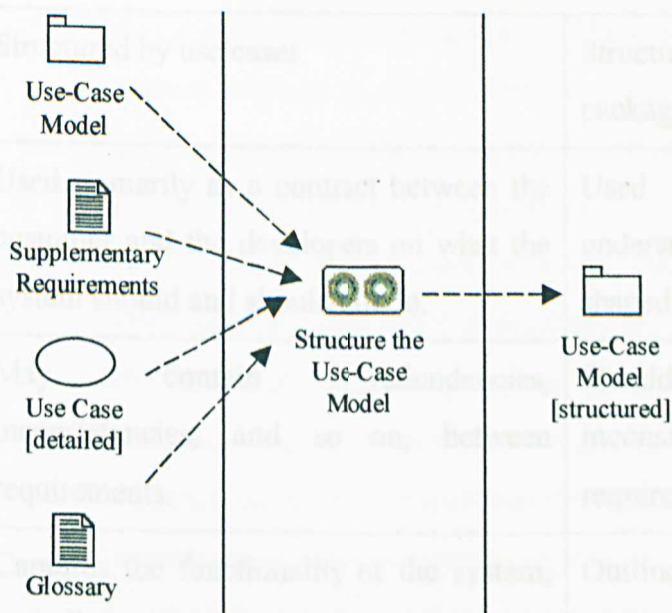


Figure 3-10 The input and result of activity structure the use-case model

3.1.1.2 Analysis

The purpose of the analysis workflow is to analyse the requirements in order to acquire a more precise understanding of the requirements and to acquire a description of the requirements that is easy to maintain and that helps us give structure to the whole system. Table 3-2 shows the comparison between the use-case model and the analysis model.

In the software life cycle (see Figure 3-4), analysis is the focus during the initial elaboration iterations. It contributes to a sound and stable architecture and facilitates an in-depth understanding of the requirements.

Figure 3-11 illustrates the workflow in analysis with the four participating activities. Each activity will be described in a subsection.

Use-Case Model (Requirements)	Analysis Model
Described using the language of the customer	Described using the language of the developer
External view of the system	Internal view of the system

Structured by use cases	Structured by stereotypical classes and packages
Used primarily as a contract between the customer and the developers on what the system should and should not do.	Used primarily by developers to understand how the system should be shaped.
May contain redundancies, inconsistencies, and so on, between requirements.	Should not contain redundancies, inconsistencies, and so on, among requirements.
Captures the functionality of the system, including architecturally significant functionality	Outlines how to realise the functionality within the system, including architecturally significant functionality.
Defines use cases that are further analysed in the analysis model.	Defines use case-realizations, each one representing the analysis of a use case from the use-case model.

Table 2 Comparison of the use-case model and the analysis model

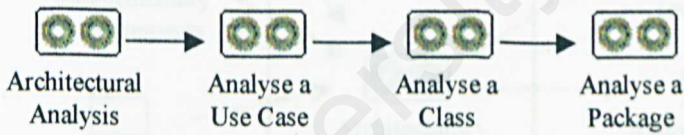


Figure 3-11 The workflow in analysis

3.1.1.2.1 Activity: Architectural analysis

The purpose of architectural analysis is to outline the analysis model and the architectural. As illustrated in Figure 3-12, this activity have as input the use-case model, the supplementary requirements, business model or domain model, and architecture description. The goal of this activity is achieved by identifying analysis packages, obvious analysis class, and common special requirements with each taking one step.

- **Identifying analysis packages.** This step is carried out based on based on functionality and problem domain. Analysis packages are obtained by allocating use cases into specific packages and then realise the corresponding

functionality within that package. These packages will localise changes to a business process, an actor's behaviour, and a set of closely related use cases. These packages are then analysed for common functionality among packages and a different package with those common functionality will be created to let other packages share the common functionality from this package. Then, the dependencies among analysis packages are defined.

- **Identifying obvious entity classes.** In this step, a preliminary proposal of the most important (architecturally significant) entity classes is prepared. These important entity classes are those that participate in use-case realisation.
- **Identifying common special requirements.** A special requirement in this context is a requirement that occurs during analysis and is important to capture so that it can be handled appropriately in the coming workflows. Then, the key characteristics of each common special requirement are identified.

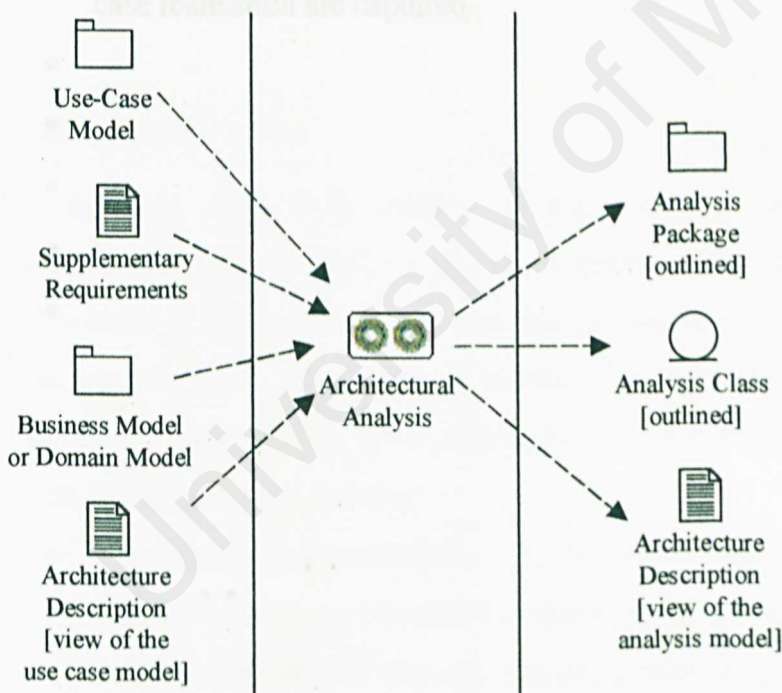


Figure 3-12 The input and result of activity architectural analysis

3.1.1.2.1 Activity: Analyse a use case

The purpose of this activity is to identify the analysis classes whose objects are needed to perform the use case's flow of events, distribute the behaviour of the use case to

interacting analysis objects, and capture special requirements on the realisation of the use case. This activity is also called use case refinement as we refine each use case as a collaboration of analysis classes. As illustrated in Figure 3-13, this activity has the use-case model, supplementary requirements, business or domain model, and architectural description of analysis model as input and produces the use case realisation of analysis and an outline of the analysis class.

- **Identifying analysis classes.** In this step, the control, entity, and boundary classes needed to realise the use case are identified and their names, responsibilities, attributes, and relationships are outlined.
- **Describing analysis object interactions.** The ways analysis objects interact are described by using collaboration diagrams that contain the participating actor instances, analysis objects, and their links.
- **Capturing special requirements.** In this step, special requirements on a use-case realisation are captured.

-
-
-
-
-
-

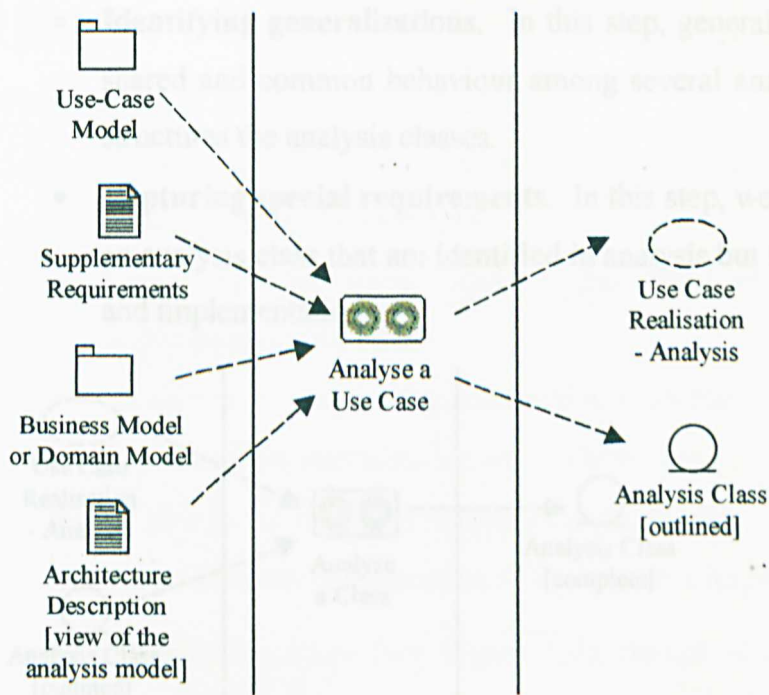


Figure 3-13 The input and result of activity analyse a use case

3.1.1.2.1 Activity: *Analyse a class*

The purposes of activity analyse a class are to identify and maintain the responsibilities of an analysis object, identify and maintain the attributes and relationships of the analysis class, and capture special requirements on the realisation of the analysis class. Hence, it will produce the complete analysis class as output as illustrated in Figure 3-14, while taking the use-case realisation of analysis and the outline of analysis class as input.

- **Identifying responsibilities.** As the name suggests, responsibilities of the analysis classes are identified in this step. The responsibilities of a class is be collected by examining all the roles that it plays in different use-case realisations.
- **Identifying attributes.** The attributes of the analysis classes will be identified in this step. Since attributes are often related to realising the responsibilities of its class, the result of the preceding step helps.
- **Identifying associations and aggregations.** In this step, associations and aggregations between classes are identified to structure them.

- **Identifying generalisations.** In this step, generalisations are used to extract shared and common behaviour among several analysis classes. This further structures the analysis classes.
- **Capturing special requirements.** In this step, we capture all requirements of an analysis class that are identified in analysis but should be handled in design and implementation.

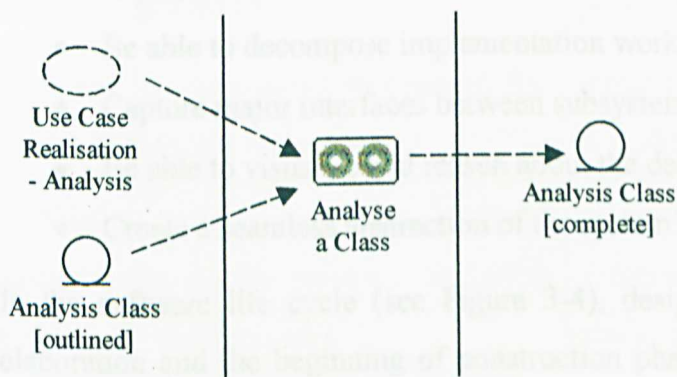


Figure 3-14 The input and result of activity analyse a class

3.1.1.2.1 Activity: Analyse a package

The purpose of this activity is to ensure that the analysis package is as independent of other packages as possible and fulfils its purpose of realising some domain classes or use cases, and to describe dependencies. This activity will produce the complete analysis package with the outline of analysis package and the architectural description as input (see Figure 3-15).

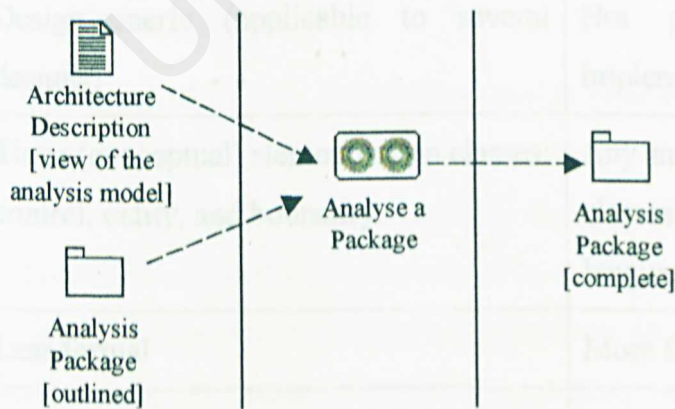


Figure 3-15 The input and result of activity analyse a package

3.1.1.3 Design

The purposes of the design workflow are to: -

- Acquire an in-depth understanding of issues regarding non-functional requirements and constraints.
- Create an appropriate input to and point of departure for subsequent implementation activities.
- Be able to decompose implementation work into more manageable pieces.
- Capture major interfaces between subsystems.
- Be able to visualise and reason about the design.
- Create a seamless abstraction of the system’s implementation.

In the software life cycle (see Figure 3-4), design is in focus during the end of elaboration and the beginning of construction phase. It contributes to a sound and stable architecture and creates a blueprint for the implementation model. Table 3-3 distinguish the design model from the analysis model. Figure 3-16 illustrates the workflow in design with the four participating activities.

Analysis Model	Design Model
Conceptual model, because it is an abstraction of the system and avoids implementation issues	Physical model, because it is a blueprint of the implementation.
Design-generic (applicable to several designs)	Not generic, but specific for an implementation
Three (conceptual) stereotypes on classes: control, entity, and boundary.	Any number of (physical) stereotypes on classes, depending on implementation language.
Less formal	More formal
Less expensive to develop	More expensive to develop
Few layers	Many layers

Dynamic, but not much focuses on sequence.	Dynamic with much focus on sequence.
Outlines the design of the system, including its architecture.	Manifests the design of the system, including its architecture.
Primarily created by “leg work,” in workshop and the like.	Primarily created by “visual programming”
May not be maintained throughout the complete software life cycle.	Should be maintained throughout the complete software life cycle.
Defines a structure that is an essential input to shaping the system – including creating the design model	Shapes the system while trying to preserve the structure defined by the analysis model as much as possible.

Table 3 Comparison of the analysis model and the design model

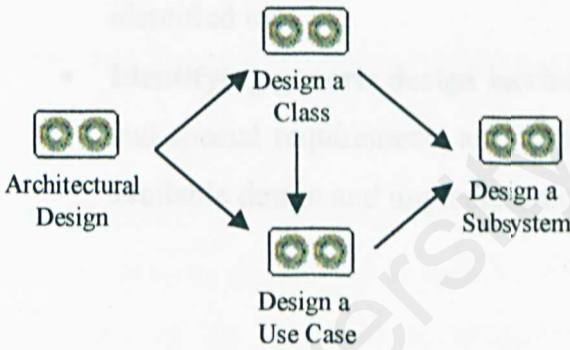


Figure 3-16 The activities in design workflow

3.1.1.2.1 Activity: Architectural design

The purpose of activity architectural design is to outline the design and development models and their architecture. The resulting sub-system interfaces, or other design elements, are then incorporated into the design model. This activity will have the use-case model, supplementary requirements, analysis model, and architecture description as input to produce an outline of subsystem, an outline of interface, an outline of design class, an outline of deployment model, and the architecture description (see Figure 3-14).

- **Identifying nodes and network configurations.** In this step, the physical network configuration will be defined. Nodes and network configuration is essential to the software's architecture.
- **Identifying subsystems and their interfaces.** In this step, subsystems are used to organise the design model into manageable pieces. Firstly, the subsystems in the application-specific and application-general layers are identified. Then, the middleware and system-software subsystems are identified. After the subsystems in all four layers are identified, dependencies among them are defined. Finally, interface for each subsystem is identified.
- **Identifying architecturally significant design class.** Architecturally significant design classes are identified at this stage to initiate the design work. These architecturally significant design classes could be identified from the architecturally significant analysis classes. Active classes that are required by the system in order to consider the concurrency requirements should be identified too.
- **Identifying generic design mechanisms.** In this step, common requirements and special requirements are studied to decide how to handle them with the available design and implementation technologies.

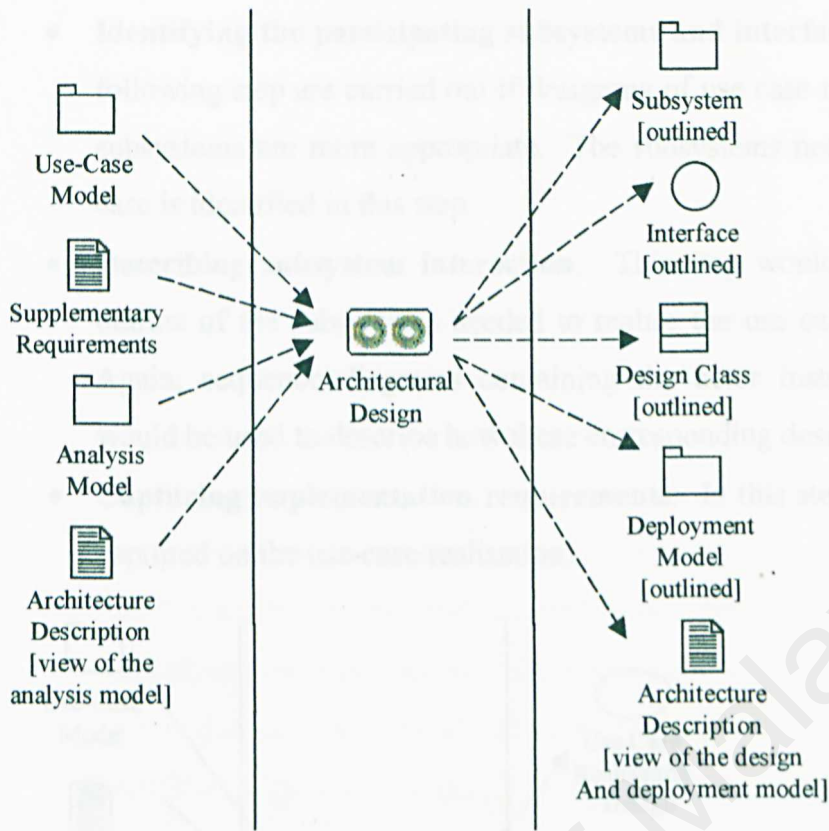


Figure 3-17 The input and result of activity architectural design

3.1.1.2.1 Activity: *Design a use case*

The purposes of this activity are to identify the design classes and/or subsystems whose instances are needed to perform the use case’s flow of events, to distribute the behaviour of the use case to interacting design objects and/or to participating subsystems, to define requirements on the operations of design classes and/or subsystems and their interfaces, and to capture implementation requirements for the use case. This step has the use-case model, supplementary requirements; analysis model, design model, and deployment model as input to produce the use case realisation of design, an outline of design classes, an outline of subsystems, and an outline of interfaces (see Figure 3-18).

- **Identifying the participating design classes.** In this step, the design classes needed to realise the use case are identified.
- **Describing design object interactions.** At this stage, an outline of the design classes needed to realise the use case is obtained. Sequence diagrams containing the actor instances and design objects are used to describe how these corresponding design objects interact.

- **Identifying the participating subsystems and interfaces.** This step and the following step are carried out if designing of use case in term of participating subsystems are more appropriate. The subsystems needed to realise the use case is identified in this step.
- **Describing subsystem interaction.** This step would be carried out if an outline of the subsystems needed to realise the use case has been identified. Again, sequence diagrams containing the actor instances and subsystems would be used to describe how these corresponding design objects interact.
- **Capturing implementation requirements.** In this step, all requirements are captured on the use-case realisation.

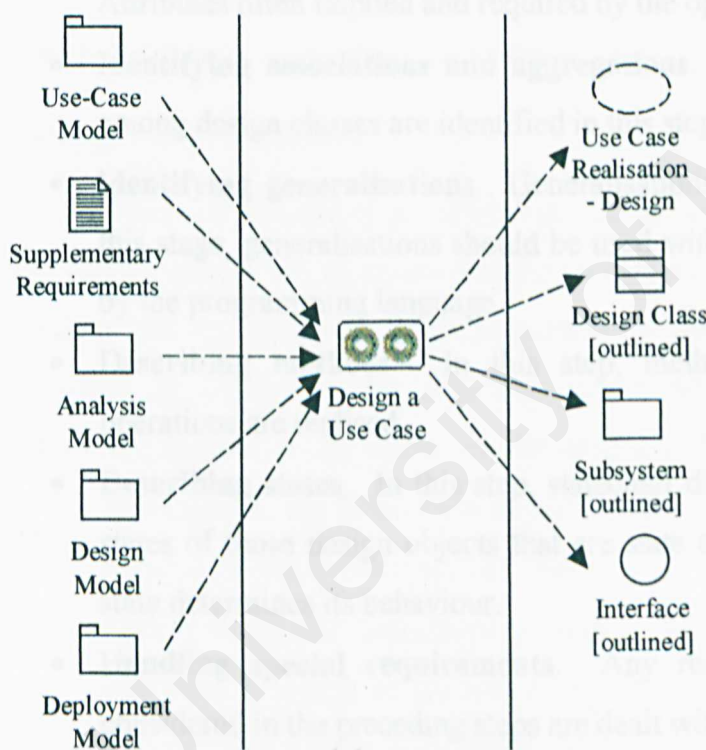


Figure 3-18 The input and result of activity design a use case

3.1.1.2.1 Activity: *Design a class*

The purpose of this activity is to create a design class that fulfils its role in use-case realisation and the non-functional requirements that apply to it. As illustrated in Figure 3-19, this activity will have as input the use-case realisation, an outline of design classes, an outline of interface, and the analysis class to produce the design class. The following are eight steps to be taken to perform this activity.

- **Outlining the design class.** The first step would be outlining the design class based on the analysis class and/or interface as input. The design classes identified here should be assigned trace dependencies to the corresponding analysis classes.
- **Identifying operations.** In this step, operations that need to be provided by the design class are identified and described using syntax of the programming language. The operations need to support all the roles the class plays in all use-case realisations it participates.
- **Identifying attributes.** As for this step, the attributes required by the design class are identified and described using syntax of the programming language. Attributes often implied and required by the operations of the class.
- **Identifying associations and aggregations.** Associations and aggregations among design classes are identified in this step.
- **Identifying generalisations.** Generalisations are identified in this step. At this stage, generalisations should be used with the same semantics as defined by the programming language.
- **Describing methods.** In this step, methods are used to specify how operations are realised.
- **Describing states.** In this step, statechart diagrams are used to describe the states of those design objects that are state controlled, which means that the state determines its behaviour.
- **Handling special requirements.** Any requirements that have not been considered in the preceding steps are dealt with in this step.

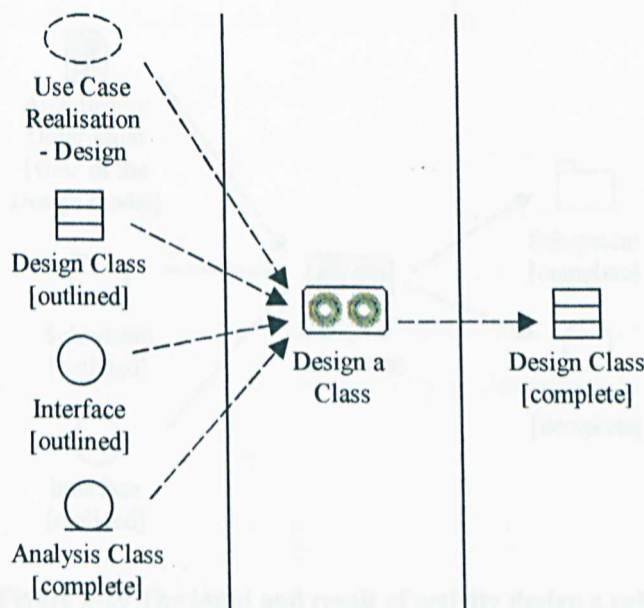


Figure 3-19 The input and result of activity design a class

3.1.1.2.1 Activity: *Design a subsystem*

The purposes of designing a subsystem are to ensure that the subsystem is as independent as possible, to ensure that the subsystem provides the right interfaces, and to ensure that the subsystem fulfils its purpose in that it offers a correct realisation of operations as defined by the interfaces. This activity would have as input the architecture description, an outline of subsystems, and an outline of interfaces.

- **Maintaining the subsystem dependencies.** In this step, dependencies among subsystems are defined and maintained.
- **Maintaining the interfaces provided by the subsystem.** In this step, interfaces provided by the subsystems are refined to ensure that it support all the roles that it plays in different use-case realisations.
- **Maintaining the subsystem contents.** Subsystem contents are maintained in this step to ensure that it fulfils its purpose by offering the correct realisation of the operations as defined by the interfaces it provides.

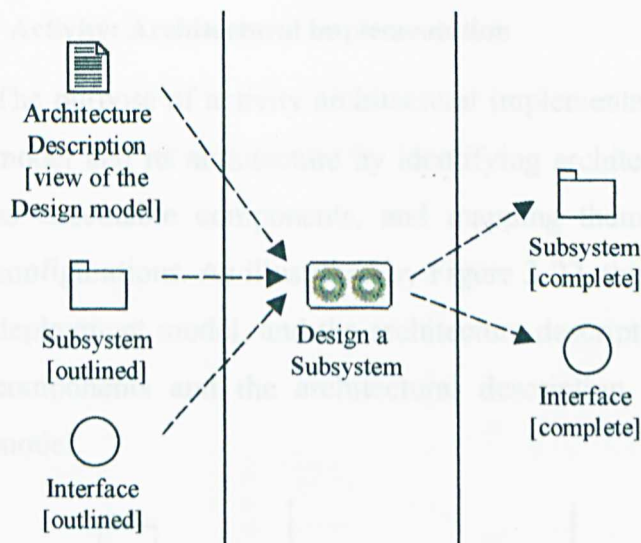


Figure 3-20 The input and result of activity design a subsystem

3.1.1.4 Implementation

The purposes of the implementation workflow are to: -

- Plan the system integrations required in each iteration.
- Distribute the system by mapping executable components onto nodes in the deployment model.
- Implement the design classes and subsystems found during design.
- Unit test the components, and then integrate them.

In the software life cycle (see Figure 3-4), implementation is the focus during the construction iterations. It is also done during elaboration phase to create executable architectural baseline and during transition phase to handle late defects. Figure 3-21 illustrates the workflow in implementation with the five participating activities.

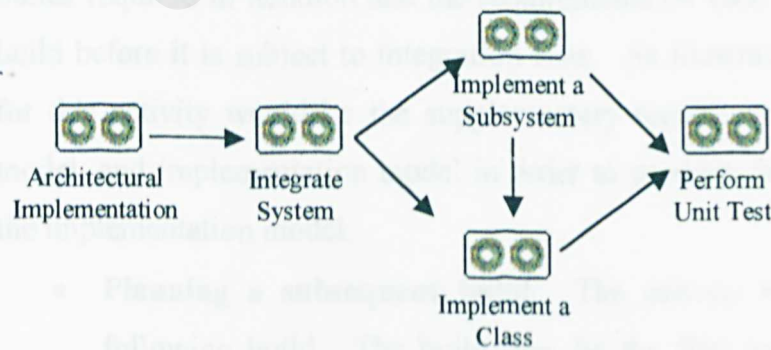


Figure 3-21 The activities in the implementation workflow

3.1.1.2.1 Activity: Architectural implementation

The purpose of activity architectural implementation is to outline the implementation model and its architecture by identifying architecturally significant components such as executable components, and mapping them to nodes in the relevant network configurations. As illustrated by Figure 3-22, this activity will have the design model, deployment model, and the architecture description as input to produce an outline of components and the architectural description of implementation and deployment model.

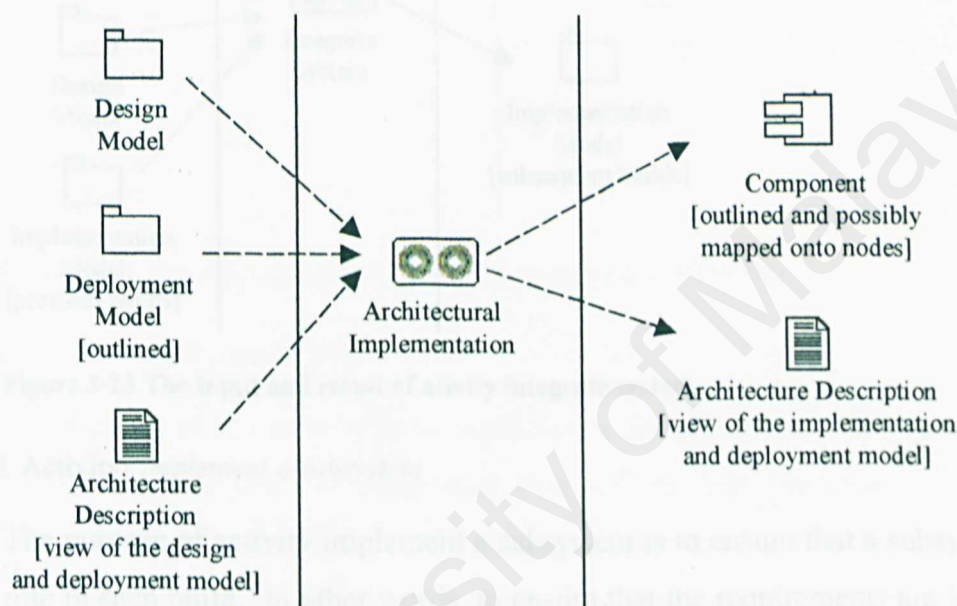


Figure 3-22 The input and result of activity architectural implementation

3.1.1.2.1 Activity: Integrate System

The purposes of this activity are to create an integration build plan describing the builds required in iteration and the requirements on each build, and to integrate each build before it is subject to integration tests. As illustrated in Figure 3-23, the input for this activity would be the supplementary requirements, use-case model, design model, and implementation model in order to produce the integration build plan and the implementation model.

- **Planning a subsequent build.** The activity will start with planning the following build. The build may be the first or continuing of the previous build. Every build should add functionality.

- **Integrating a build.** In this step, builds are integrated. It is done by collecting the implementation subsystems and components, compiling them, and linking them into a build.

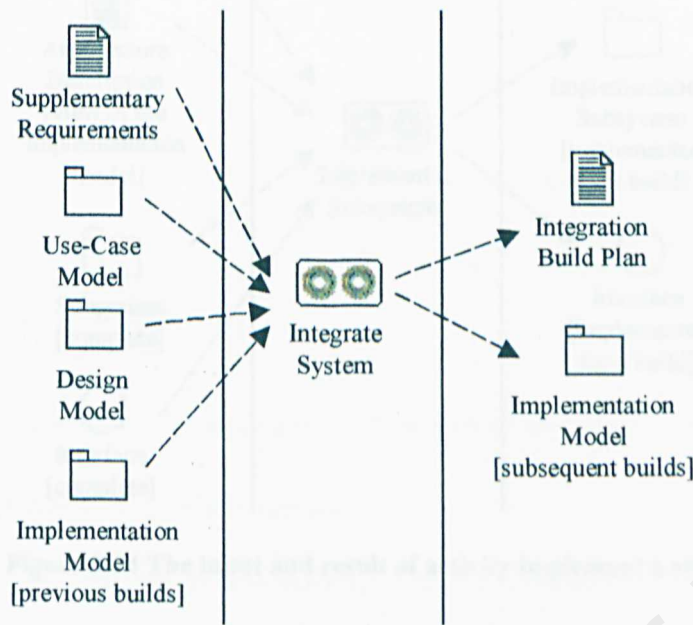


Figure 3-23 The input and result of activity integrate system

3.1.1.2.1 Activity: *Implement a subsystem*

The purpose of activity implement a subsystem is to ensure that a subsystem fulfils its role in each build. In other words, to ensure that the requirements are implemented in the build and those that affect the subsystem are correctly implemented by components or other subsystem within the subsystem. The input for this activity is the integration builds plan, architectural description, design subsystem, and interface to produce the implementation subsystem, and the interface (see Figure 3-24).

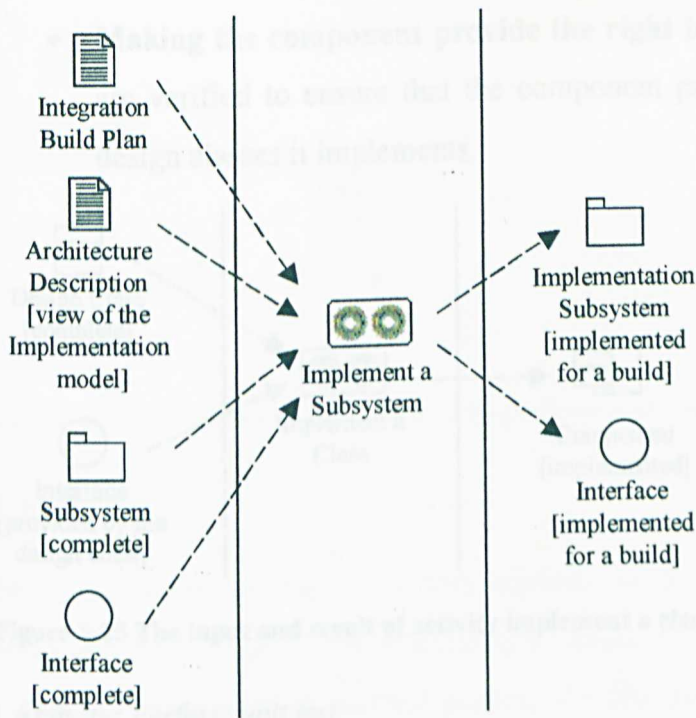


Figure 3-24 The input and result of activity implement a subsystem

3.1.1.2.1 Activity: *Implement a Class*

The purpose of this activity is to implement a design class in a file component, which includes outlining a file component that will contain the source code, generating source code from the design class, implementing the operations of the design class, and ensuring that the components provide the same interface as the design class. As illustrated in Figure 3-25, design class, and interface provided by the design class will form the input and the activity produce the implemented component as the result. The activity will be carried out in four steps.

- **Outlining the file components.** In this step, the file components where the source code that implements the design class resides are outlined.
- **Generating code from a design class.** The source code is generated according to the design class in this step. If the design class has been described using the syntax of the programming language during design, this step will be straightforward.
- **Implementing operations.** In this step, operations of the class will be produced. This step involves choosing a suitable algorithm and specified data structure, and then coding the actions required but eh algorithm.

- **Making the component provide the right interface.** In this step, interfaces are verified to ensure that the component provide the same interface as the design classes it implements.

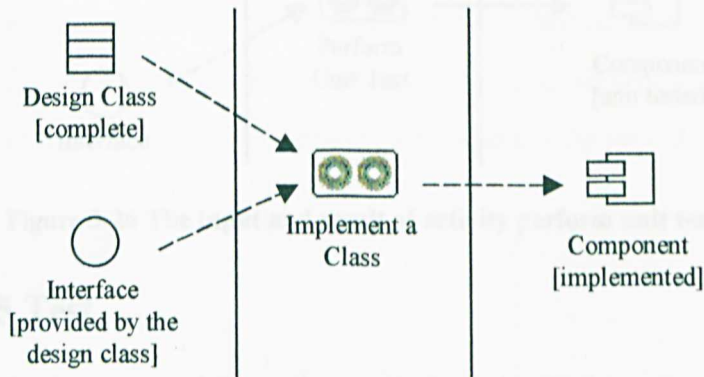


Figure 3-25 The input and result of activity implement a class

3.1.1.2.1 Activity: *Perform unit test*

The purpose of this activity is to test the implemented components as individual units. The component and interface will form the input of this activity to produce the unit-tested component (see Figure 3-26). The following are two types of unit test performed.

- **Specification tests** Specification test is also known as “black-box test”. It is done to verify the component’s behaviour without considering how that behaviour is implemented within the component. The test is conducted by observing the output the component will return when given certain input and when starting in a particular state.
- **Structure tests** Structure test is also know as “white-box test”. It is done to verify that a component works internally as intended. During structure testing, all code should be test. In other words, every statement has to be executed at least once.

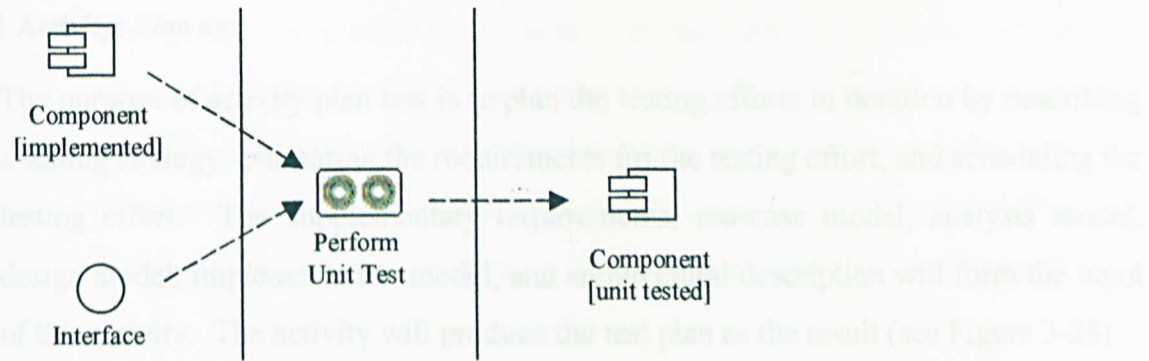


Figure 3-26 The input and result of activity perform unit test

3.1.1.5 Test

In the test workflow, the result from implementation are verified by testing each build including both internal and intermediate builds, as well as the final versions of the system to be released to external parties. As illustrated in Figure 3-27, this workflow will be performed with six activities.

The purposes of this workflow are to plan the tests required in each iteration, design and implement the tests by creating test cases that specify what to test, and perform the various tests and handle the results of each test.

In the software life cycle (see Figure 3-4), test is in focus during the elaboration phase, when the executable architectural baseline is tested, and during construction, when the bulk of the system is implemented. However, some initial test plan may occur during the inception phase when the system is scoped.

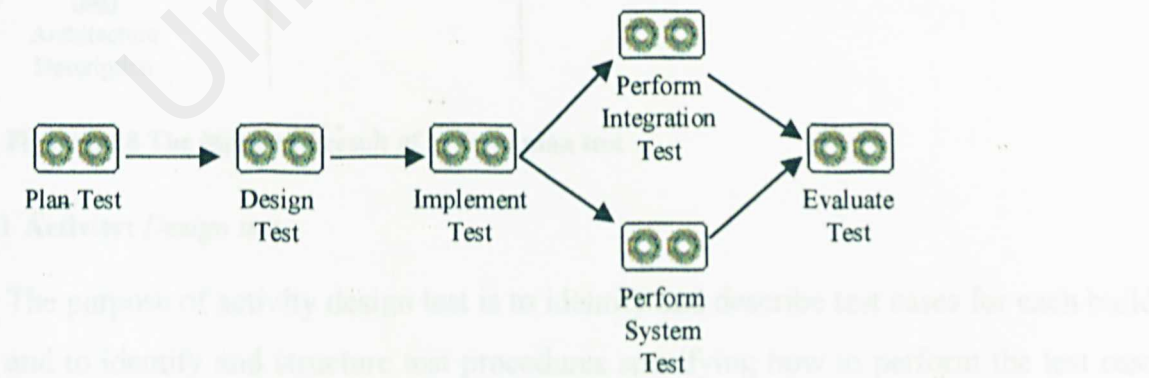


Figure 3-27 The activities in workflow test

3.1.1.2.1 Activity: Plan test

The purpose of activity plan test is to plan the testing efforts in iteration by describing a testing strategy, estimating the requirements for the testing effort, and scheduling the testing effort. The supplementary requirements, use-case model, analysis model, design model, implementation model, and architectural description will form the input of this activity. The activity will produce the test plan as the result (see Figure 3-28).

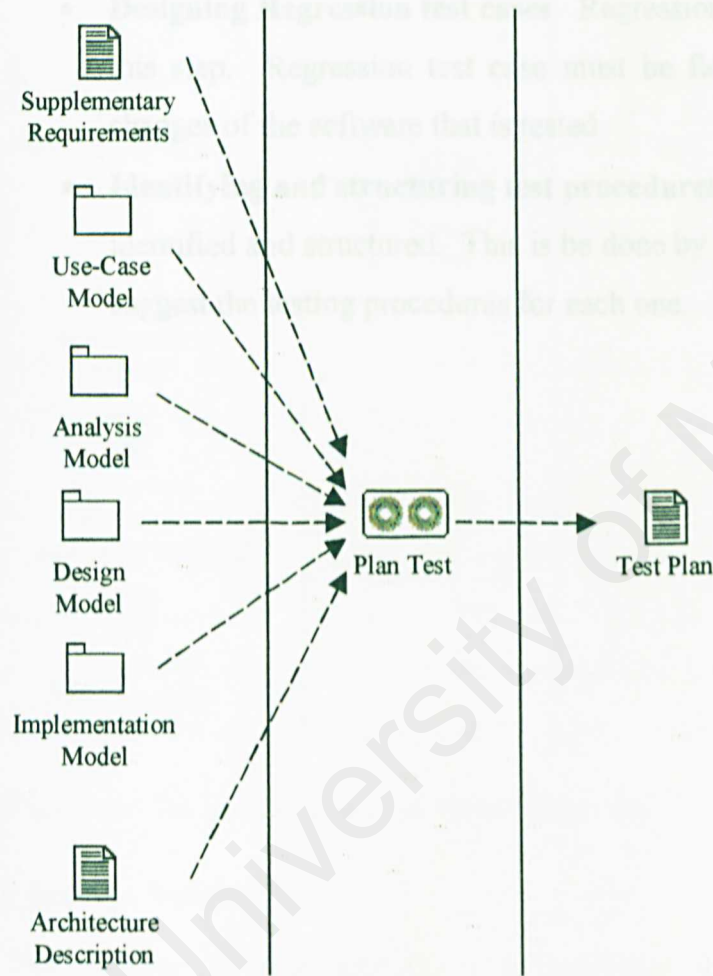


Figure 3-28 The input and result of activity plan test

3.1.1.2.1 Activity: Design test

The purpose of activity design test is to identify and describe test cases for each build, and to identify and structure test procedures specifying how to perform the test case. As illustrated in Figure 3-29, the input for this activity is the supplementary requirements, use-case model, analysis model, design model, implementation model, architectural description, and test plan to produce the test case and test procedure.

- **Designing integration test cases.** In this step, integration test case is designed. Integration test cases are used to verify that the components interact properly with each other after they have been integrated into a build.
- **Designing system test cases.** System test cases will be designed in this step. System test cases are used to verify that the system functions properly as a whole.
- **Designing Regression test cases.** Regression test cases would be designed in this step. Regression test case must be flexible enough to be resilient to changes of the software that is tested.
- **Identifying and structuring test procedures.** In this step, test procedures are identified and structured. This is be done by working through the test cases to suggest the testing procedures for each one.

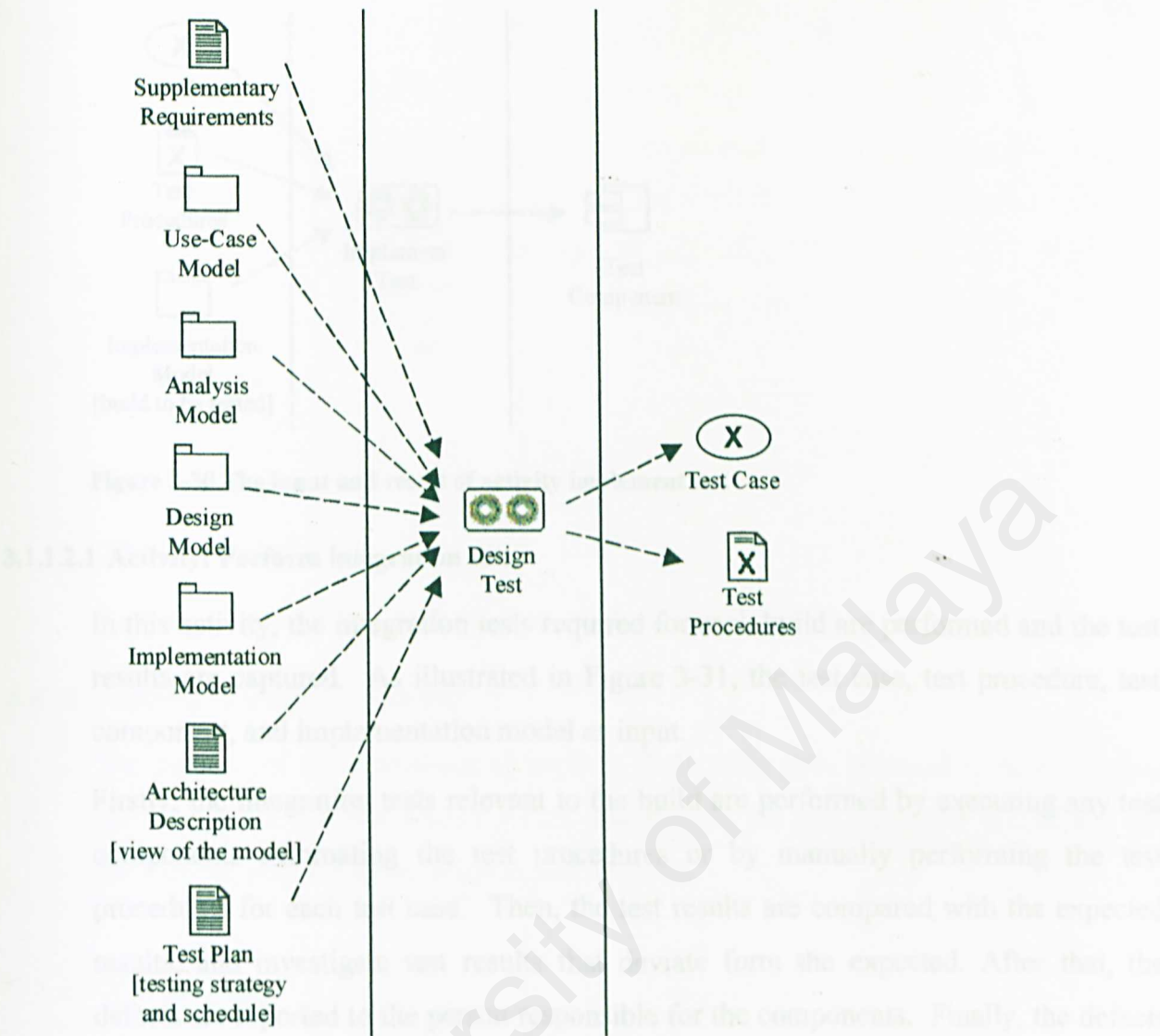


Figure 3-29 The input and result of activity design test

3.1.1.2.1 Activity: *Implement test*

The purpose of this activity is to automate test procedures by creating test components. However, not all test procedures can be automated. This activity will use test case, test procedures, and implementation model as input and produces the test component (see Figure 3-30).

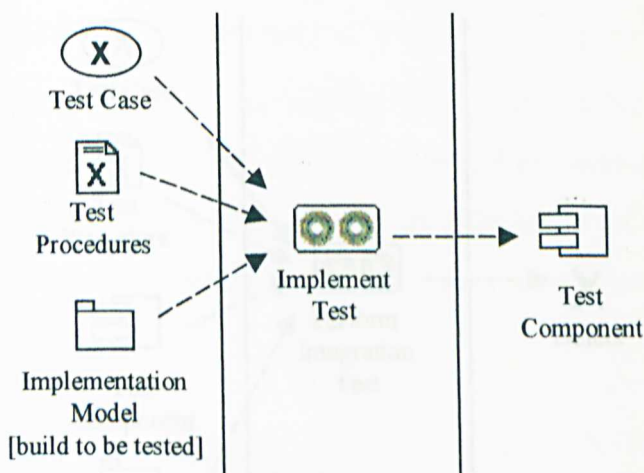


Figure 3-30 The input and result of activity implement test

3.1.1.2.1 Activity: Perform integration test

In this activity, the integration tests required for each build are performed and the test results are captured. As illustrated in Figure 3-31, the test case, test procedure, test component, and implementation model as input.

Firstly, the integration tests relevant to the build are performed by executing any test components automating the test procedures or by manually performing the test procedures for each test case. Then, the test results are compared with the expected results, and investigate test results that deviate form the expected. After that, the defects are reported to the person responsible for the components. Finally, the defects are reported for the purpose of evaluating the overall results of the testing effort.

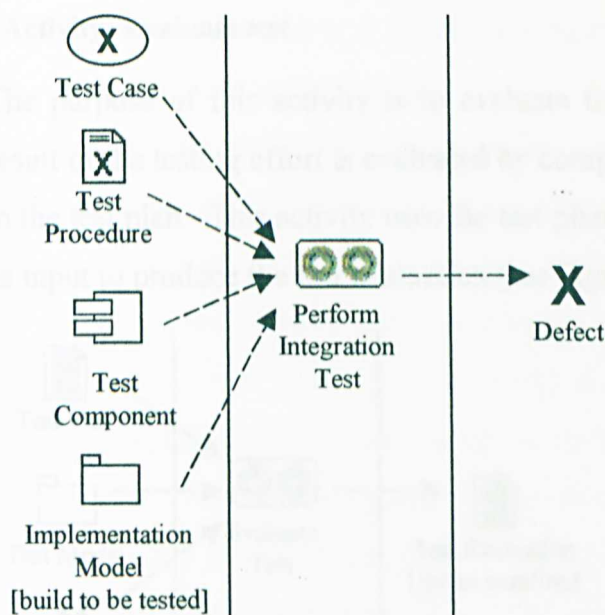


Figure 3-31 The input and result of activity perform integration test

3.1.1.2.1 Activity: Perform system test

The purpose of this activity is to perform the system tests required in each iteration and to capture the test results. This activity has the test case, test procedure, test component, and implementation model as input.

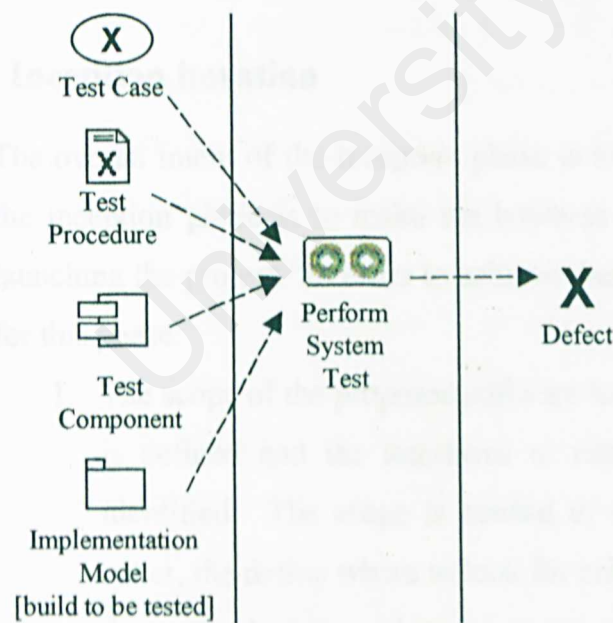


Figure 3-32 The input and result of activity perform system test

3.1.1.2.1 Activity: Evaluate test

The purpose of this activity is to evaluate the testing efforts within iteration. The result of the testing effort is evaluated by comparing the results with the goals outlined in the test plan. This activity uses the test plan, the test model, and the defect reported as input to produce the test evaluation (see Figure 3-33).

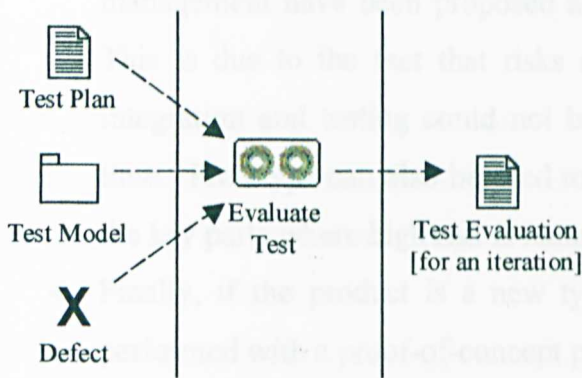


Figure 3-33 The input and result of activity evaluate test

3.1.2 The Iteration Workflows

This section will describe the four iteration workflows with emphases of each iteration workflow on the activities in the five core workflows.

3.1.2.1 Inception iteration

The overall intent of the inception phase is to launch the project. Hence, the goal in the inception phase is to make the business case to the extent necessary to justify launching the project. In order to achieve these goals, there are four steps to be taken for this phase.

1. The scope of the proposed software has to be delimited. The system boundary is defined and the interfaces to related systems outside the boundary are identified. The scope is needed to understand what the architecture has to cover, the define where to look for critical risks, and to provide the boundaries for cost, schedule, and return-on-investment estimates.
2. The candidate architecture of the system has to be described or outlined. The emphases are on those parts that are new, risky, or difficult. This step ends with an architecture description and no executable prototype is built since the

goal of this phase is to assure that a stable architecture could be created to support the system scope.

3. Critical risks have to be identified. In this phase, the stress is on risks that affect feasibility, meaning risks that threaten the successful development of the system. Other risks are recorded for consideration in later phases. Risk management has been proposed at this early stage to avoid project failure. This is due to the fact that risks discovered at late stages such as system integration and testing could not be mitigated within budget and scheduled time. Prototype can also be used to manage and mitigate risk by prototyping the key parts where high risk is identified.
4. Finally, if the product is a new type of software, a demonstration may be performed with a proof-of-concept prototype.

3.1.2.2 Elaboration iteration

The primary product of the elaboration phase is a stable architecture.

1. An architectural baseline that covers the architecturally significant functionality of the system is created. This architectural baseline will consist of the model artefacts, architecture description, and executable implementation. Hence, it takes the architecture a step from the inception phase by creating the executable architecture.
2. Significant risks, that is, risks that could upset the plans and schedule of later phase, are identified.
3. The levels to be attained by quality attributes are specified.
4. Use cases to about eighty percent of functional requirements are captured. This would be sufficient to plan for the construction phase.
5. A proposal covering all the resources is prepared.

3.1.2.3 Construction iteration

The general objective of this phase is a product with initial operational capability. This phase ends with a product ready for beta testing. The general activities of this phase include:

1. Extending the use-case identification, description, and realisation to the entire body of the use cases.
2. Finishing analysis, design, implementation, and test
3. Maintaining the integrity of the architecture.
4. Monitoring critical and significant risks carried over from the first two phases.

3.1.2.4 Transition iteration

This phase typically begins with the beta release. This signifies that the software product is capable of initial operations and is distributed to a representative sample of the community of actual user. The activities of this phase include:

- Preparation activities.
- Advising the customer on updating the environment in which the software is to operate.
- Preparation of manuals and other documentation for product release.
- Adjusting the software to operate under the actual parameters of the user environment.
- Correcting defects found after feedback from the beta tests.
- Modifying the software in the light of unforeseen problems.

3.2 Strength of the proposed approach

The Unified Software Development Process is the outcome of more than thirty years of experience [1]. The methodology is shaped in a way that has solves the many problems that many software development methods possess.

Use case model for requirements capture: The two main concerns of requirements capture are to find the true requirements and to represent them in a suitable way. Use case model has been the choice of the Unified Process. In a use-case model, there are use cases, which represent a piece of functionality in the system, and actors, which represent the users and any external system that the system interacts with.

According to Karl Wiegner, “the perspective provided by use cases reinforces the ultimate goal of software engineering: to create products that let customers do useful

work.” This is due to the fact that use-case model states the requirements with focus on value added to the user. In other words, requirement capture is according to the perspective of each type of user, considering what the system should provides in order for them to do their work.

Use cases have been adopted almost universally for capturing the requirements of software systems in general but of component-based system in particular.

Use-case driven: To be use-case driven means that a development process proceeds through a series of workflows that are initiated from the use cases. In other words, they drive the whole development process. Figure 3-34 illustrates the models of the Unified Process where all the models have dependencies with the use-case model. In other words, use cases are traceable through all the models.

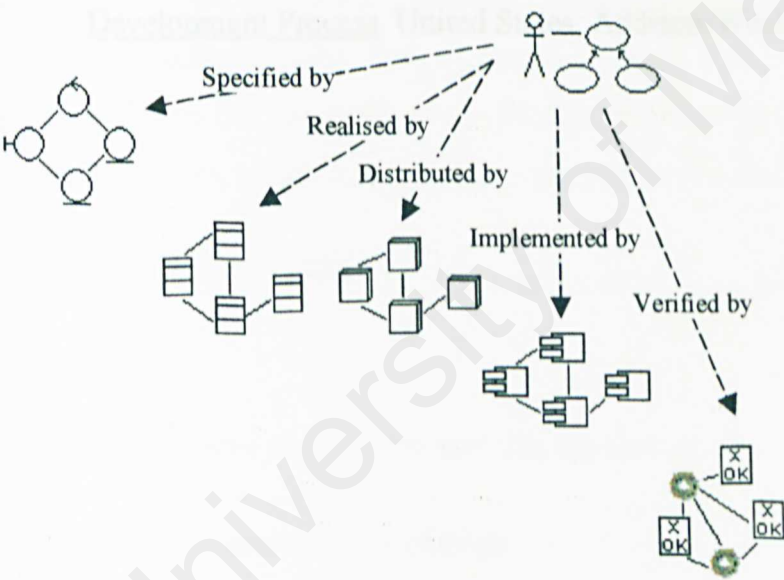


Figure 3-34 Models of the Unified Process

Iterative and incremental: Iterative and incremental development enables developing software in small steps with major and minor milestones with which the developer could control the development better.

It can be shaped to deal with the development of any kinds of software product due to the flexibility of the iteration workflow. For larger projects, the construction phase could be carried out in more iteration while for complex and green field projects, the

inception and elaboration phase could be extended to more iteration to better understand the project before making further steps.

3.3 Chapter Summary

This chapter has provides a description of the Unified Software Development Process, the methodology in for this project. The chapter also specified the strength of the Unified Process. Among the strength is the use of use case model for requirement capture, the process being use case driven, and the development using the iteration and incremental model.

3.4 References

[1] Jacobson, I., Booch, G., Rumbaugh, J.(1999).The Unified Software Development Process. United States: Addison-Wesley.

Chapter 4 Requirements Capture and Analysis

This chapter presents the result of requirements capture and analysis on the requirements. The chapter consist of four sections. Section 4.1 presents the result of requirements capture presented as use cases and their flow of events. As for the result of analysis, it is presented in Section 4.2 with use-case realisation of the analysis model and the analysis classes. The final two sections, Section 4.3 and Section 4.4 would summarise the chapter and states the references respectively.

4.1 Requirements Capture

This section presents the outcome of requirements capture in three subsections. The first subsection provides a description of actors and use cases identified and the flow of events of these use cases. Statechart diagrams will also be included for use cases that have a more complex flow of events. The second subsection provides the user interface design for this application. Finally, the third subsection will state the whole use case diagram, which forms the requirements of the product.

4.1.1 Actor and Use Cases

Actor: User

A User represents a person who uses this application to perform calculation.

Use Case: Perform General Calculation

This use case is used by the User to perform all supported expression-based calculation.

Precondition: The calculation mode is in General Calculation window is opened.

Flow of events

Basic Path

1. The User invokes the use case by opening the General Calculation window.
2. The User keys in the calculation expression with the keys on the keypad or with their equivalent accelerators. The applications append the token referred

by the key pressed to the calculation expression and refresh the calculation expression line on the screen.

- 3. The User decided to acquire the answer and press the equal key on the keypad. The application evaluates the calculation expression and provides the answer if there is no error. After the answer is displayed, the application will go back to step two.

Alternative Paths

In step two, if the General Calculation window closed or the application is terminated, the use-case instance terminates.

In step three, if there is error in the calculation expression, the equivalent error message will be provided on the answer line of the display. After the error message is displayed, the application will go back to step two.

Postcondition: The use-case instance ends when the application is terminated or the calculation mode has changed.

Figure 4-1 illustrates a statechart diagram describing the use case Perform General Calculation.

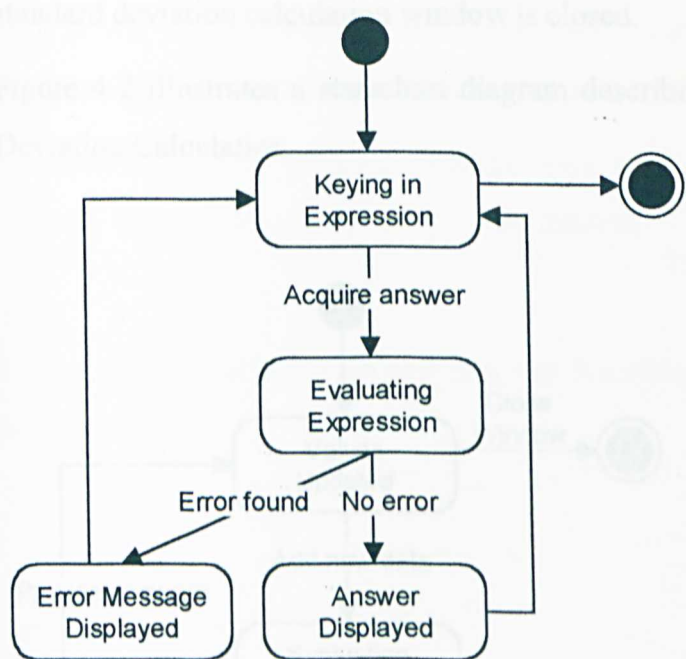


Figure 4-1 Statechart diagram showing the states of use case Perform General Calculation

Use Case: Perform Standard Deviation Calculation

This use case is used by the User to perform standard deviation calculation.

Precondition: The standard deviation window is opened.

Flow of events

Basic Path

- 4. The User invokes the use case by bringing out the standard deviation window.
- 5. The User keys in the data for standard deviation calculation into a list. The application will evaluate the list and provide the values of number of data, mean, population standard deviation, sample standard deviation, sum of values, and sum of squares as new data is entered.

Alternative Paths

In step two, if the standard deviation window is closed, the use-case instance terminates.

Postcondition: The use-case instance ends when the application is terminated or the standard deviation calculation window is closed.

Figure 4-2 illustrates a statechart diagram describing the use case Perform Standard Deviation Calculation.

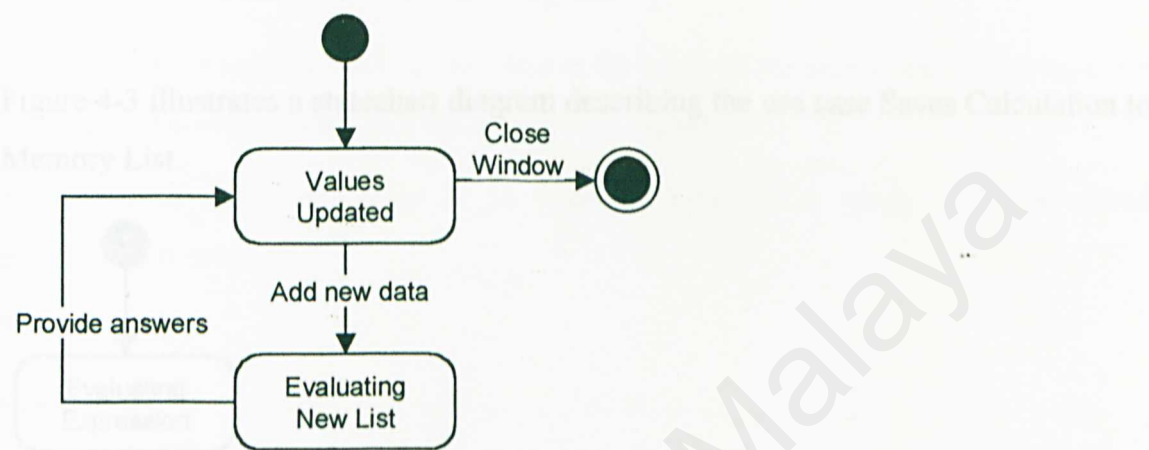


Figure 4-2 Statechart diagram of use case Perform Standard Deviation Calculation

Use Case: Saves Calculation to Memory List.

This use case is used by the User to preserve calculations performed for later use by saving them into the memory list.

Precondition: The calculator is in General Calculation mode and a calculation has been performed.

Flow of events

Basic Path

1. The User invokes the use case by pressing the save button on the memory list window. The application evaluates the expression and produces the answer if there is no error. The application saves the calculation and its answer to the list.
2. The use-case instance terminates.

Alternative Paths

In step one, if there is error in the expression, the equivalent error message will be produced. The application will not save it in the memory list.

Postcondition: The use-case instance ends when the calculation is saved into the memory list or there is error with the expression.

Figure 4-3 illustrates a statechart diagram describing the use case Saves Calculation to Memory List.

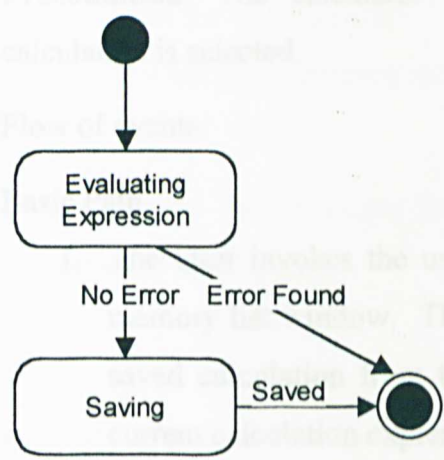


Figure 4-3 Statechart diagram showing the states of use case Saves Calculation to Memory List

Use Case: Retrieve Calculation from Memory List

This use case is used by the User to retrieve the calculations that has been saved into the memory list.

Precondition: The calculator is in General Calculation mode and one saved calculation is selected.

Flow of events

Basic Path

1. The User invokes the use case by pressing the retrieve calculation button on the memory list window. The application retrieved the calculation from the memory list and overrides the current calculation expression with this calculation.
2. The use-case instance terminates.

Postcondition: The use-case instance ends when the calculation is retrieved from the memory list to the current calculation expression.

Use Case: Retrieve value from the memory list

This use case is used by the User to retrieve the value of the saved calculation from the memory list and appended into the calculation expression.

Precondition: The calculator is in General Calculation mode and one saved calculation is selected.

Flow of events

Basic Path

1. The User invokes the use case by pressing the retrieve value button on the memory list window. The application retrieved the slot name of the selected saved calculation from the memory list and appended the slot name to the current calculation expression.
2. The use-case instance terminates.

Postcondition: The use-case instance ends when the slot name of the selected saved calculations appended to the current calculation expression.

Use Case: Delete calculation from the memory list

This use case is used by the User to delete the saved calculation from the memory list.

Precondition: The calculator is in General Calculation mode and one saved calculation is selected.

Flow of events

Basic Path

1. The User invokes the use case by pressing the delete value button on the memory list window. The application deletes the selected stored calculation from the memory list.
2. The use-case instance terminates.

Postcondition: The use-case instance ends when the selected saved calculation is deleted from memory list.

Use Case: Retrieve Calculation from History List

This use case is used by the User to retrieve the calculations from the history list.

Precondition: The calculator is in General Calculation mode and one calculation in the history list is selected.

Flow of events

Basic Path

1. The User invokes the use case by pressing the retrieve calculation button on the history list window. The application retrieved the calculation from the history list and overrides the current expression with this calculation.
2. The use-case instance terminates.

Postcondition: The use-case instance ends when the calculation is retrieved from the history list to the current calculation expression.

Use Case: Retrieve Constant from Constant List

This use case is used by the User to retrieve the constant to be used in calculation.

Precondition: The calculator is in General Calculation mode and one constant is selected.

Flow of events

Basic Path

1. The User invokes the use case by pressing the retrieve button on the constant list window. The application retrieved the constant symbol from the constant list and appends it to the current expression.
2. The use-case instance terminates.

Postcondition: The use-case instance ends when the constant symbol is retrieved from the constant list and appended to the current calculation expression.

4.1.2 User Interface

This subsection specifies the user interface design of the use cases.

Figure 4-4 illustrates the user interface design of use case Perform General Calculation. The screen for this window will use a custom-made component for expression-based calculation. There is a toolbar above the screen for activating the Standard Deviation Calculation window, the memory list window, the history list window, and the constant list window.

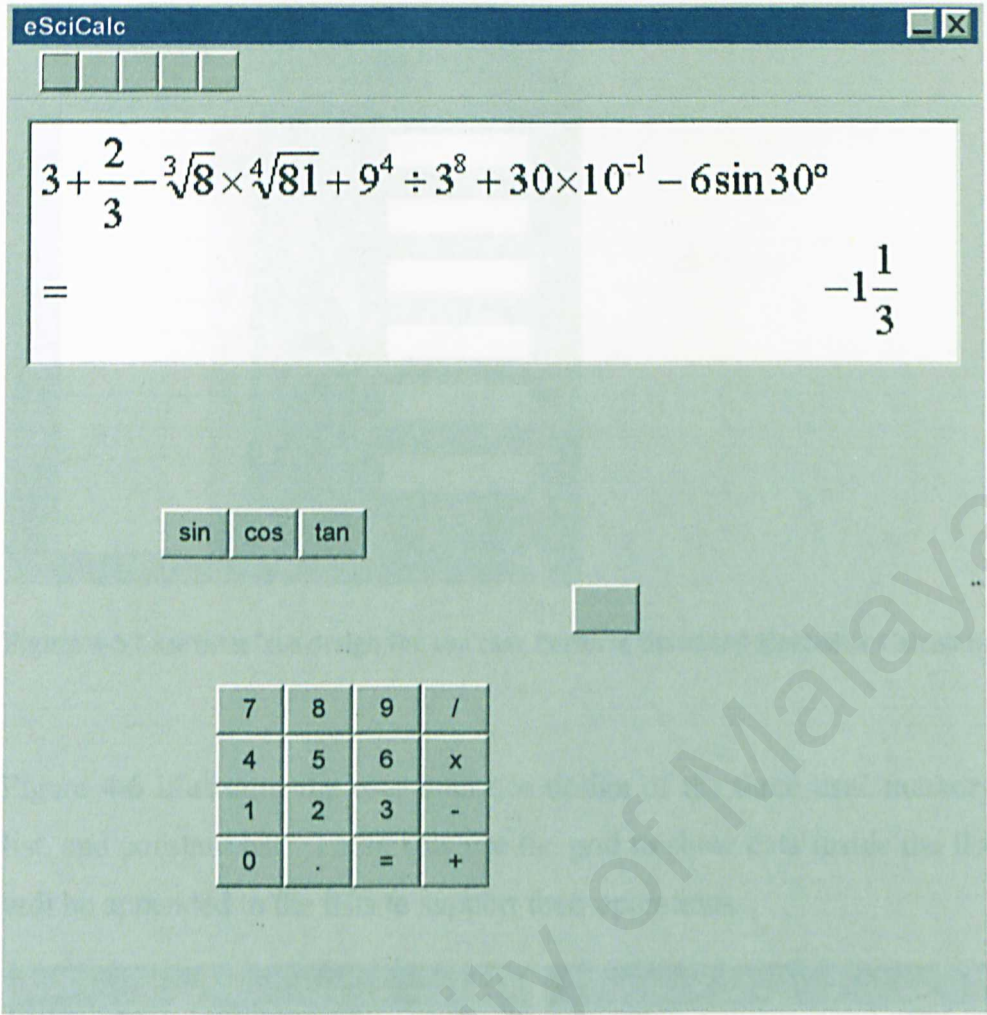


Figure 4-4 User interface design for use case Perform General Calculation

The user interface design of use case Perform Standard Deviation Calculation is illustrated in Figure 4-5. A list will be used for add and delete of data. As the content of the list change, the values of the six variables at the right side will be recalculated with the new list.

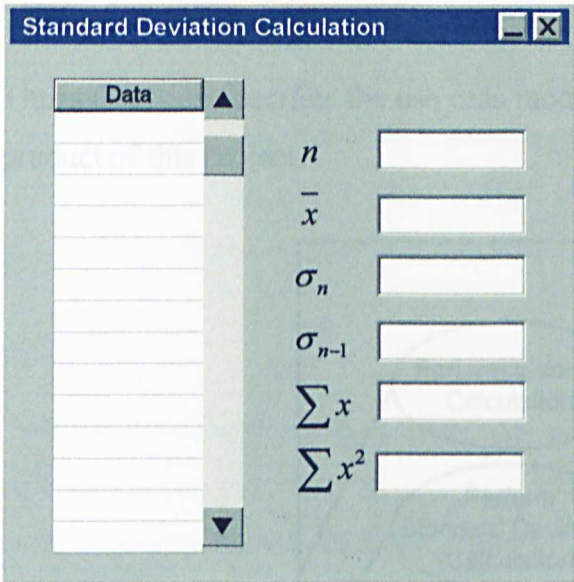


Figure 4-5 User interface design for use case Perform Standard Deviation Calculation

Figure 4-6 illustrates the user interface design of the three lists, memory list, history list, and constant list. These lists use the grid to show data inside the list. A tool bar will be appended to the lists to support their operations.

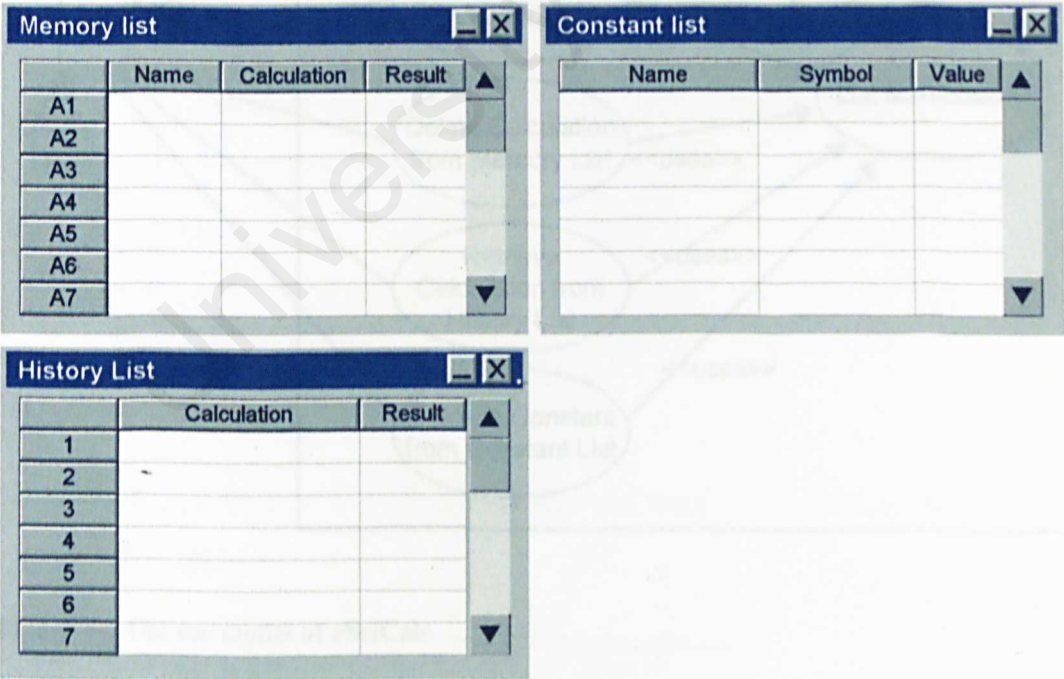


Figure 4-6 User interface design for the four lists

4.1.3 Use-Case Model

This subsection specifies the use case model, which expresses the requirements of the product of this project.

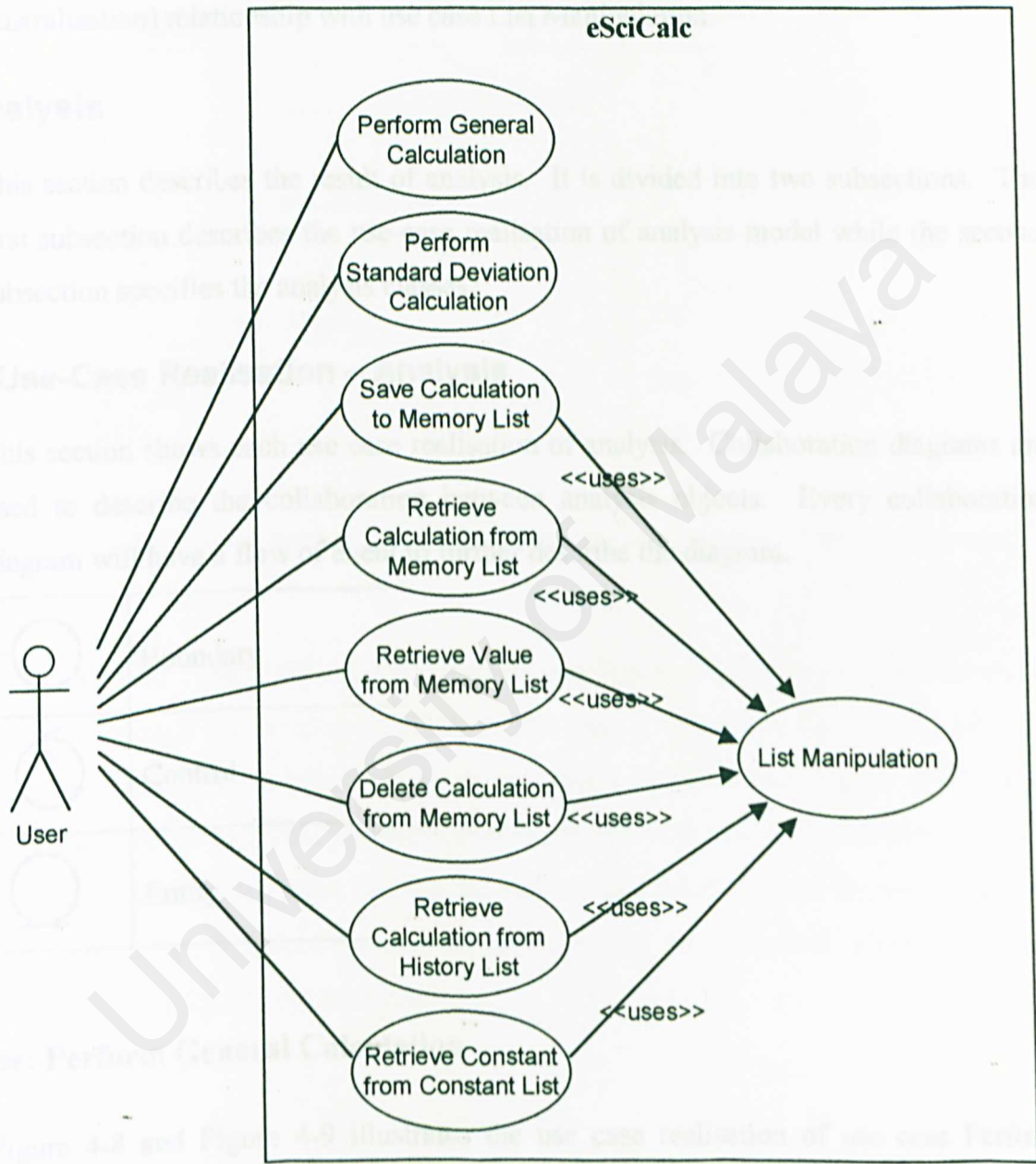


Figure 4-7 Use case model of eSciCalc

Figure 4-7 illustrates the structured use-case model of application eSciCalc. The user interacts with eight use cases: Perform General Calculation, Perform Standard Deviation Calculation, Save Calculation to Memory List, Retrieve Calculation from




Memory List, Retrieve Value from Memory List, Delete Calculation from Memory List, Retrieve Calculation from History List, and Retrieve Constant from Constant List. The flow of events of all these use cases has been specified in Section 4.1.1. As illustrated in Figure 4-7, the bottom six use cases have a uses (equivalent to generalisation) relationship with use case List Manipulation.

4.2 Analysis

This section describes the result of analysis. It is divided into two subsections. The first subsection describes the use-case realisation of analysis model while the second subsection specifies the analysis classes.

4.2.1 Use-Case Realisation – Analysis

This section shows each use case realisation of analysis. Collaboration diagrams are used to describe the collaboration between analysis objects. Every collaboration diagram will have a flow of event to further describe the diagram.

	Boundary
	Control
	Entity

Use Case: Perform General Calculation

Figure 4-8 and Figure 4-9 illustrates the use case realisation of use case Perform General Calculation.

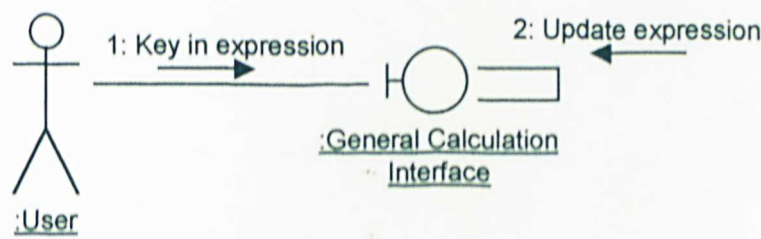


Figure 4-8 Collaboration diagram describing key-in part of use case Perform General Calculation

Flow of event: The user keys in the calculation expression (see Figure 4-8 step 1) and the application updates the expression displayed (see Figure 4-8 step 2).

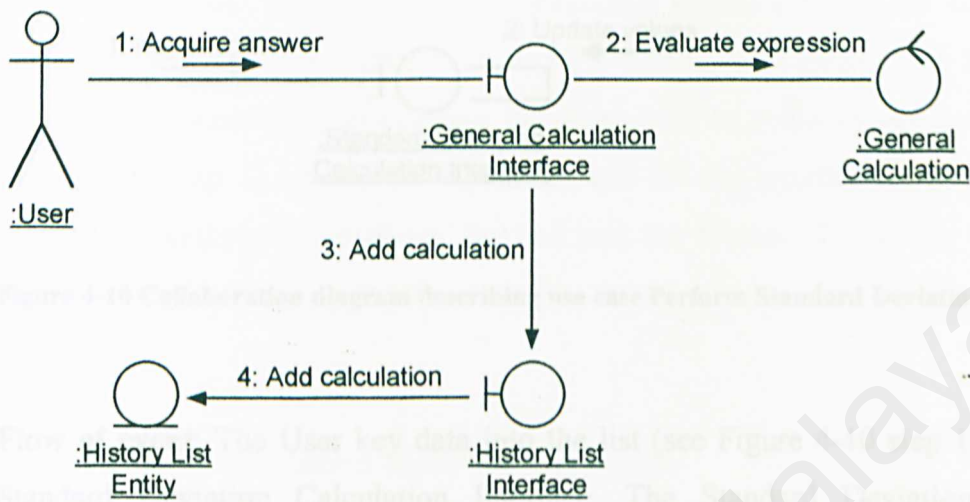


Figure 4-9 Collaboration diagram describing the execute expression part of the use case model

Flow of event: The User acquires answer (see Figure 4-9 step 1) to the expression keyed in through the General Calculation Interface. The General Calculation Interface uses the General Calculation object to evaluate expression (see Figure 4-9 step 2). With the result at hand, the General Calculation Interface updates the display and adds calculation to the history list through the History List Interface (see Figure 4-9 step 3). The History List Interface updates the displayed list and adds this calculation to the database using the History List Entity.

Use Case: Perform Standard Deviation Calculation

Figure 4-10 illustrates the use case realisation of use case Perform Standard Deviation Calculation.

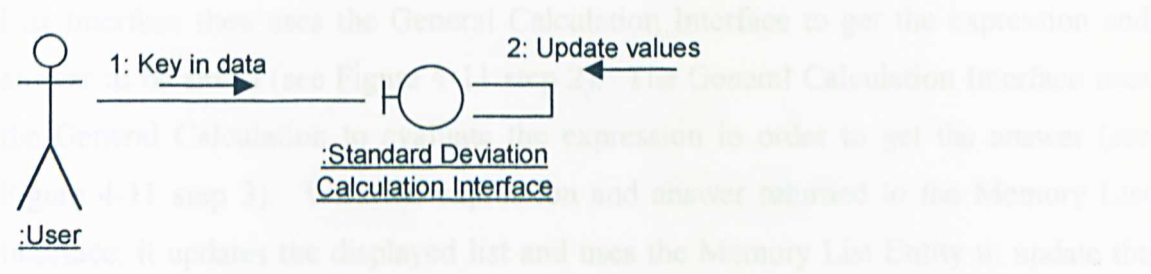


Figure 4-10 Collaboration diagram describing use case Perform Standard Deviation Calculation

Flow of event: The User key data into the list (see Figure 4-10 step 1) through the Standard Deviation Calculation Interface. The Standard Deviation Calculation Interface object then updates all the values with the new data as input (see Figure 4-10 step 2). Every time a modification to the data list is performed the values will be updated.

Use Case: Save Calculation to Memory List

Figure 4-11 illustrates the use case realisation of use case Save Calculation to Memory List.

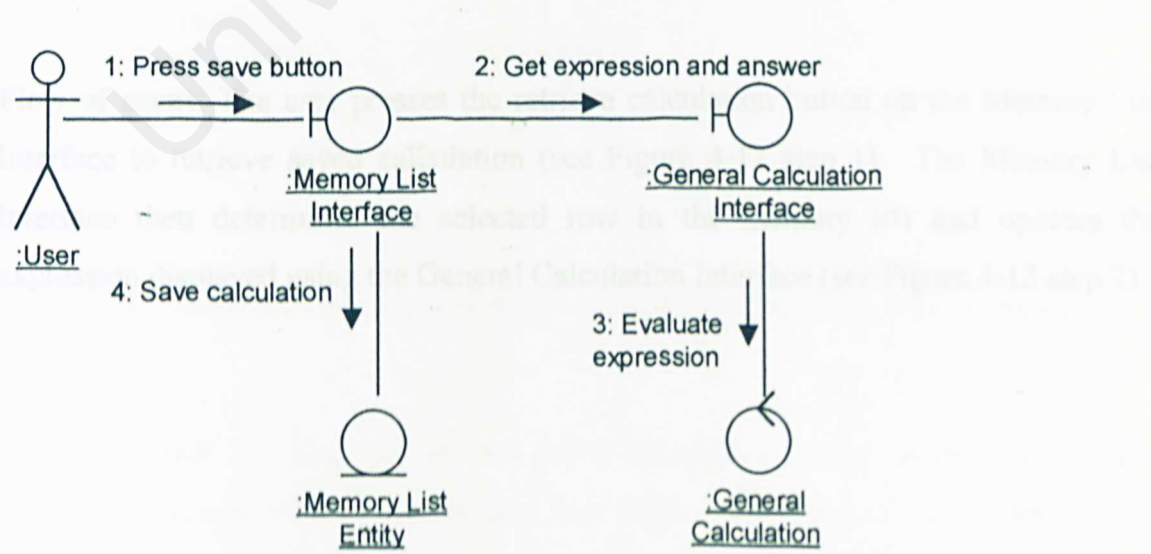


Figure 4-11 Collaboration diagram describing use case Saves Calculation to Memory List

Flow of event: The User presses the save button on the Memory List Interface (see Figure 4-11 step 1) to save the current calculation into the memory list. The Memory List Interface then uses the General Calculation Interface to get the expression and answer to be saved (see Figure 4-11 step 2). The General Calculation Interface uses the General Calculation to evaluate the expression in order to get the answer (see Figure 4-11 step 3). With the expression and answer returned to the Memory List Interface, it updates the displayed list and uses the Memory List Entity to update the database (see Figure 4-11 step 4).

Use Case: Retrieve Calculation from Memory List

Figure 4-12 illustrates the use case realisation of use case Retrieve Calculation from Memory List.

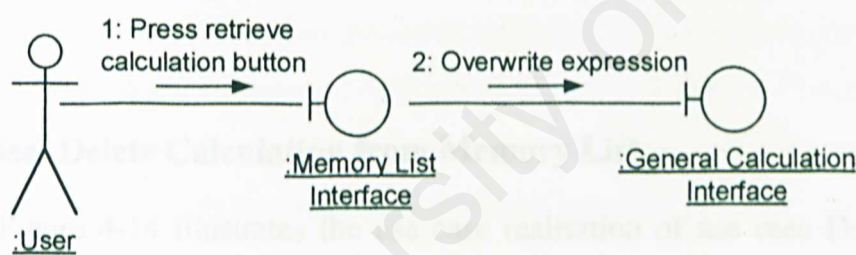


Figure 4-12 Collaboration diagram describing use case Retrieve Calculation from Memory List

Flow of event: The user presses the retrieve calculation button on the Memory List Interface to retrieve saved calculation (see Figure 4-12 step 1). The Memory List Interface then determines the selected row in the memory list and updates the expression displayed using the General Calculation Interface (see Figure 4-12 step 2).

Use Case: Retrieve Value from Memory List

Figure 4-13 illustrates the use case realisation of use case Retrieve Value from Memory List.

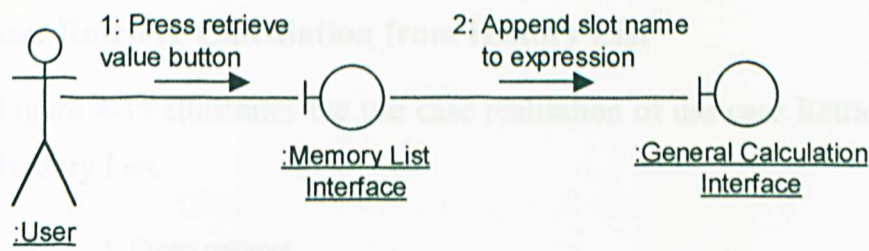


Figure 4-13 Collaboration diagram describing use case Retrieve Value from Memory List

Flow of event: The User press retrieve value button on the Memory List Interface to retrieve saved value (see Figure 4-13 step 1). The Memory List Interface then determines the selected row in the memory list and appends the slot name to the expression using the Get Calculation Interface (see Figure 4-13 step 2).

Use Case: Delete Calculation from Memory List

Figure 4-14 illustrates the use case realisation of use case Delete Calculation from Memory List.

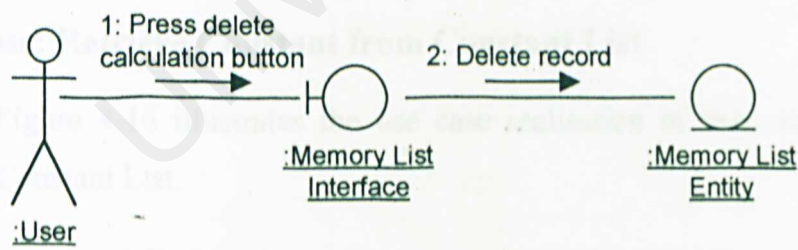


Figure 4-14 Collaboration diagram describing use case Delete Calculation from Memory List

Flow of event: The User presses the delete calculation button on the Memory List Interface to delete a stored calculation (see Figure 4-14 step 1). The Memory List Interface then determines the selected row in the memory list and deletes the

equivalent record from the database using the Memory List Entity (see Figure 4-14 step 2) and updates the displayed list.

Use Case: Retrieve Calculation from History List

Figure 4-15 illustrates the use case realisation of use case Retrieve Calculation from History List.

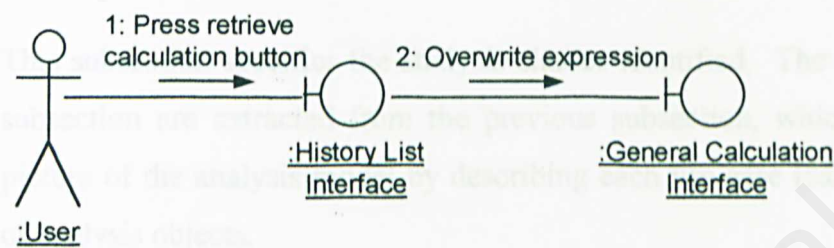


Figure 4-15 Collaboration diagram describing use case Retrieve Calculation from History List

Flow of event: The User press retrieve calculation button on the History List Interface to retrieve any of the ten latest calculations (see Figure 4-15 step 1). The History List Interface then determines the selected row in the history list and overwrites the current expression using the General Calculation Interface (see Figure 4-15 step 2).

Use Case: Retrieve Constant from Constant List

Figure 4-16 illustrates the use case realisation of use case Retrieve Constant from Constant List.

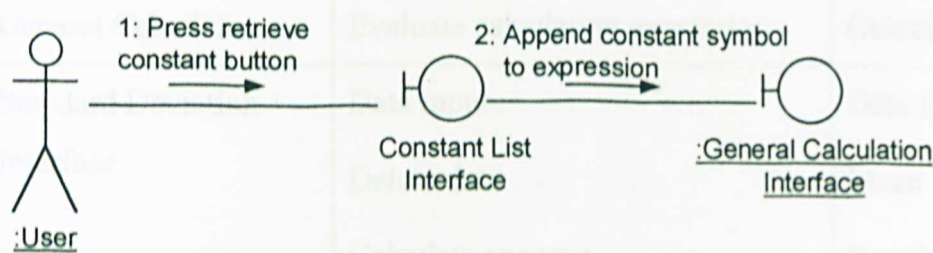


Figure 4-16 Collaboration diagram describing use case Retrieve Constant from Constant List

Flow of event: The User press retrieve value button on the Constant List Interface to retrieve constant for calculation (see Figure 4-16 step 1). The Constant List Interface then determines the selected row in the constant list and appends the constant symbol to the current expression with the General Calculation Interface (see Figure 4-16 step 2).

4.2.2 Analysis Class

This subsection specifies the analysis classes identified. The results presented in this subsection are extracted from the previous subsection, which has provided a clear picture of the analysis model by describing each use case realisation as collaboration of analysis objects.

The responsibilities and attributes of each analysis classes are described in Table 4-1. These responsibilities are needed by the analysis classes to perform use case realisation.

Analysis Class	Responsibilities	Attributes
General Calculation Interface	Update display Append token Overwrite expression Provide expression and answer Get answer	Calculation expression Answer
General Calculation	Evaluate calculation expression	Calculation expression
Standard Deviation Interface	Data input Delete data Calculate answers	Data list Mean Population Sample Number of data

		Sum of values Sum of Squares
Memory List Interface	Refresh list Save calculation Provide calculation retrieval Provide value retrieval Delete calculation Get the whole list	Memory list
Memory List Entity	Add record Provide whole list retrieval Delete record	Slot number Name Calculation Value
History List Interface	Refresh list Retrieve calculation Add calculation Get the whole list	History list
History List Entity	Add record Provide whole list retrieval Delete record	Slot number Calculation expression
Constant List Interface	Refresh list Get the whole list Provide constant retrieval	Constant list
Constant List Entity	Provide whole list retrieval	Constant symbol Constant value

Table 4 Analysis classes with their responsibilities and attributes

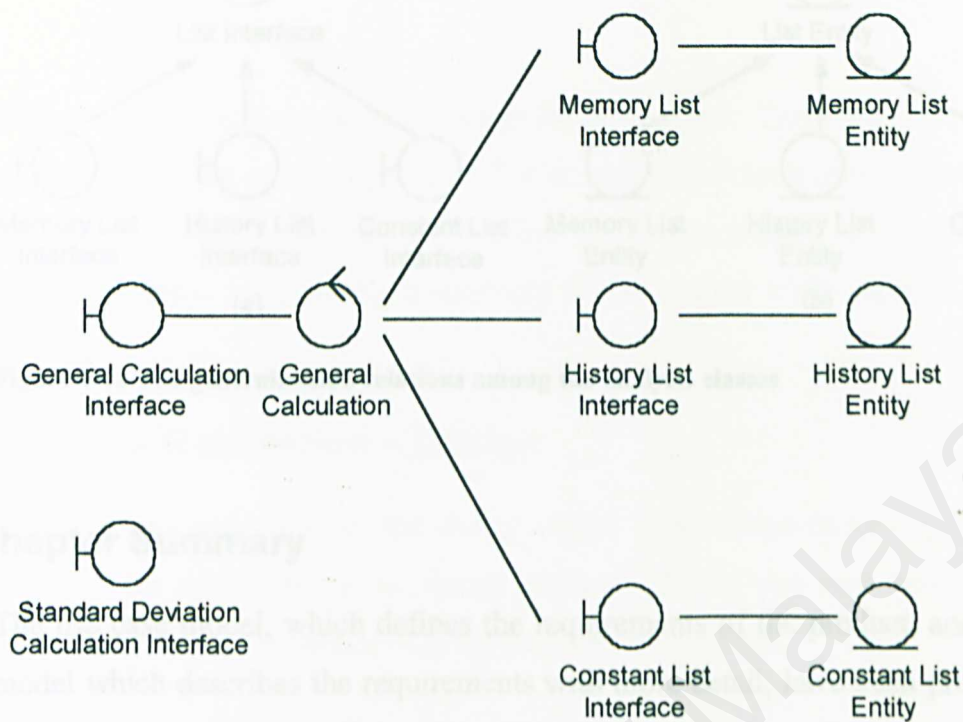


Figure 4-17 Class diagram showing the analysis classes of eSciCalc

Figure 4-17 illustrates the analysis classes of this project using class diagram. The classes are mainly divided into two groups, one for each calculation modes. The standard deviation mode are realised with just one boundary class. The second group consists of analysis classes that realises the general calculation mode including the three lists to assist calculation.

Figure 4-18 illustrates the generalisation relationships that involved the analysis classes. The generalisation relationships are excluded from Figure 4-17 to reduce complexity and crowdedness.

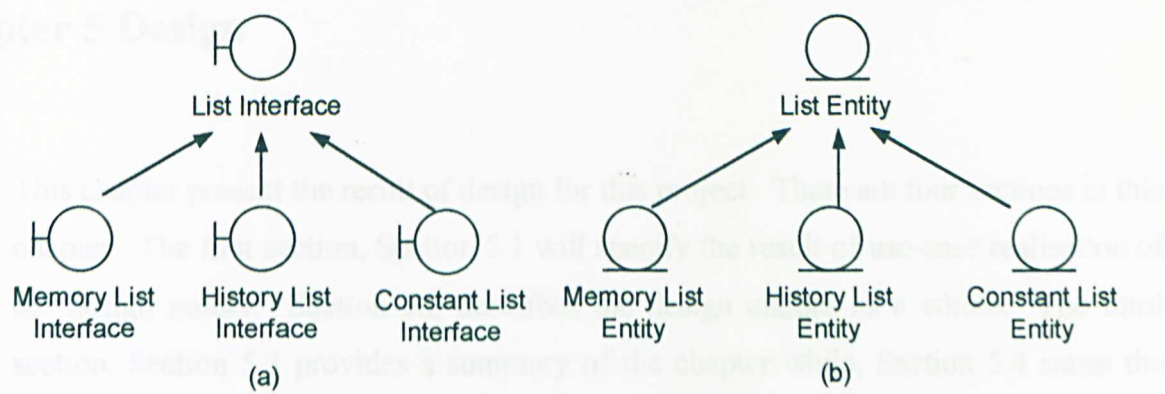


Figure 4-18 The generalisation relations among the analysis classes

5.1 Use-Case Realisation – Design

4.3 Chapter Summary

The use case model, which defines the requirements of the product, and the analysis model which describes the requirements with more detail, have been presented in this chapter. Nine analysis classes have been identified to realise the eight use cases identified earlier.

4.4 References

[1] Jacobson, I., Booch, G., Rumbaugh, J. (1999). The Unified Software Development Process. United States: Addison-Wesley.

[2] Jacobson, I., Booch, G., Rumbaugh, J. (1999). The Unified Modelling Language User Guide. United States: Addison-Wesley.

Chapter 5 Design

This chapter presents the result of design for this project. There are four sections in this chapter. The first section, Section 5.1 will specify the result of use-case realisation of the design model. Section 5.2 describes the design classes as a whole. The third section, Section 5.3 provides a summary of the chapter while, Section 5.4 states the references.

5.1 Use-Case Realisation – Design

The use case realisation of the design model is described using sequence diagrams showing the collaboration of design objects. Every use case realisation will be described with one sequence diagram and the flow of event.

Use Case: Perform General Calculation

Figure 5-1 illustrates the use-case realisation for use case Perform General Calculation with a sequence diagram.

Flow of event: The user keys in the calculation statement by repetitively pressing the keys on the GenKeypad panel. When a key is pressed, GenKeypad uses the keyPressed() function to inform GenCalcUIMan of the key being pressed. The GenCalcUIMan will append the token of the equivalent key pressed to the calculation statement and refreshes the GenScreen control with refreshScreen() function.

When the user has decided to acquire the answer for the expression keyed in, the equal key on the GenKeypad panel is pressed. The GenKeypad then send the key pressed information to GenCalcUIMan with the keyPressed() function. The GenCalcUIMan uses the function getAnswer() to acquire answer from GenCalc. GenCalc will evaluate the expression and return the answer to GenCalcUIMan. After that, the GenCalcUIMan refreshes the screen with the answer using the function refreshScreen(). Next, the GenCalcUIMan add the calculation to the history list using the addCalculation(). The HistListUIMan then update the record in the database using function updateRecord() in the HistGridEntity and update the displayed list with function updateList() in HistGrid.



Figure 5.1 Singapore Airport describing use and function General Aviation

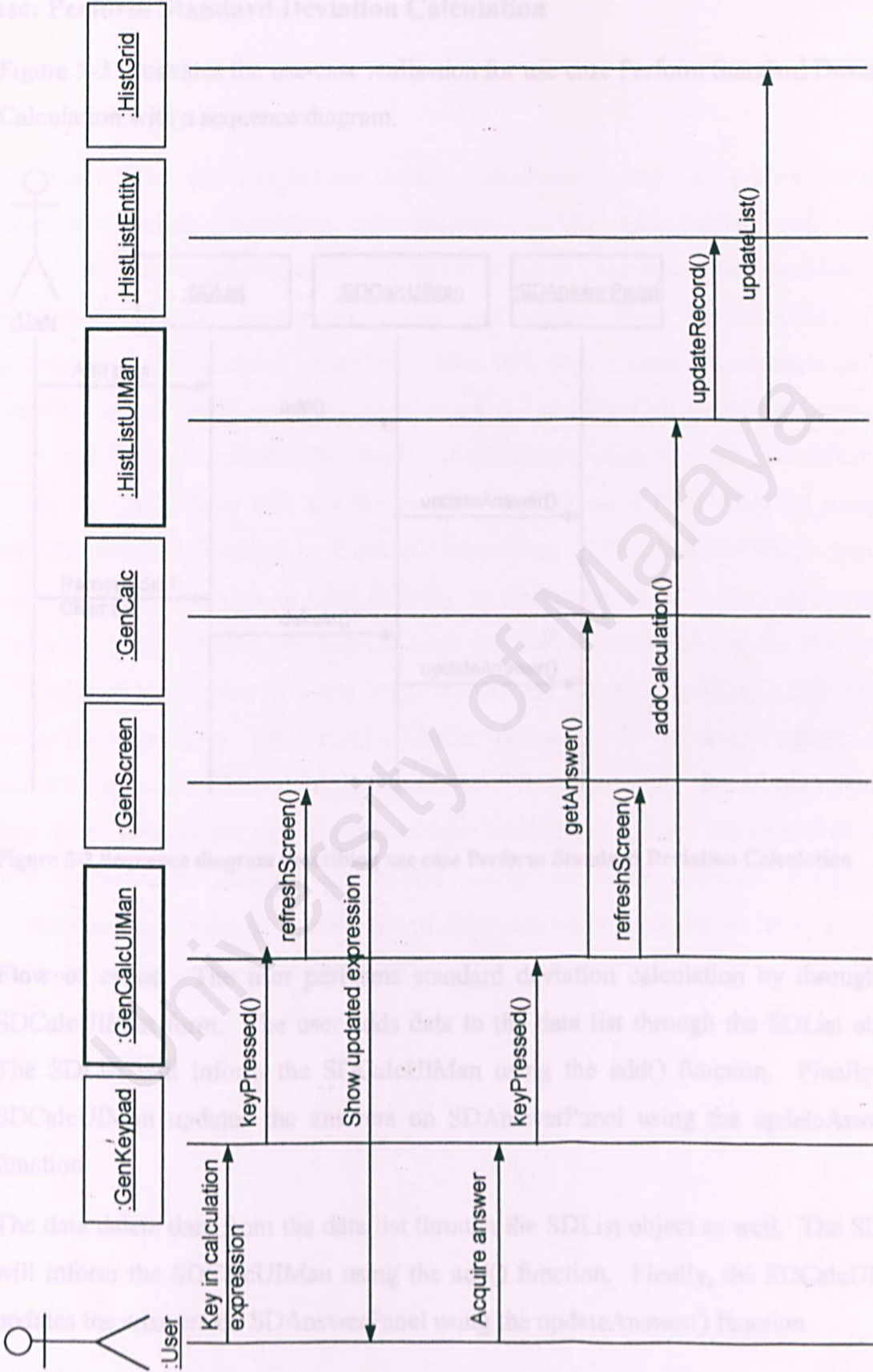


Figure 5-1 Sequence diagram describing use case Perform General Calculation

Use Case: Perform Standard Deviation Calculation

Figure 5-2 illustrates the use-case realisation for use case Perform Standard Deviation Calculation with a sequence diagram.

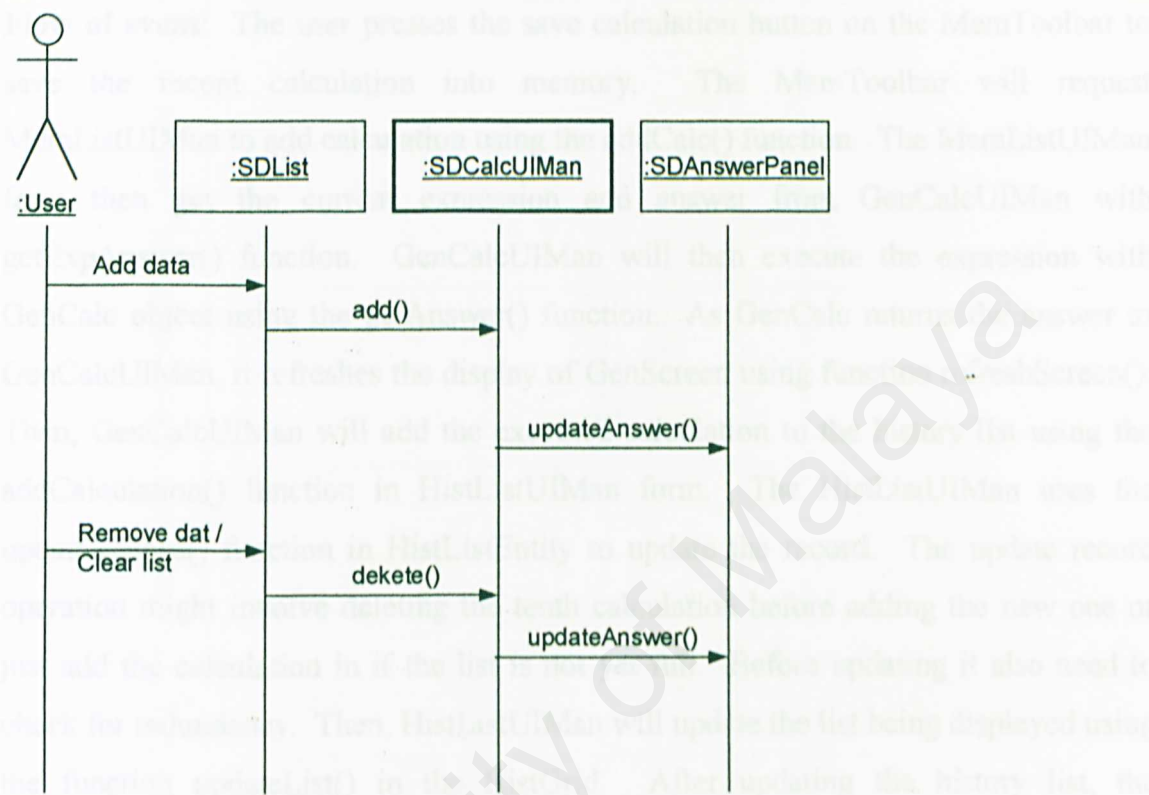


Figure 5-2 Sequence diagram describing use case Perform Standard Deviation Calculation

Flow of event: The user performs standard deviation calculation by through the SDCalcUIMan form. The user adds data to the data list through the SDList object. The SDList will inform the SDCalcUIMan using the add() function. Finally, the SDCalcUIMan updates the answers on SDAnswerPanel using the updateAnswer() function.

The data delete data from the data list through the SDList object as well. The SDList will inform the SDCalcUIMan using the add() function. Finally, the SDCalcUIMan updates the answers on SDAnswerPanel using the updateAnswer() function.

Use Case: Save Calculation to Memory List

Figure 5-3 illustrates the use-case realisation for use case Save Calculation to Memory List with a sequence diagram.

Flow of event: The user presses the save calculation button on the MemToolbar to save the recent calculation into memory. The MemToolbar will request MemListUIMan to add calculation using the addCalc() function. The MemListUIMan form then get the current expression and answer from GenCalcUIMan with getExpAnswer() function. GenCalcUIMan will then execute the expression with GenCalc object using the getAnswer() function. As GenCalc returns the answer to GenCalcUIMan, it refreshes the display of GenScreen using function refreshScreen(). Then, GenCalcUIMan will add the executed calculation to the history list using the addCalculation() function in HistListUIMan form. The HistListUIMan uses the updateRecord() function in HistListEntity to update the record. The update record operation might involve deleting the tenth calculation before adding the new one or just add the calculation in if the list is not yet full. Before updating it also need to check for redundancy. Then, HistListUIMan will update the list being displayed using the function updateList() in the HistGrid. After updating the history list, the expression and answer that MemListUIMan wanted would have been received and it will update the memory list record using the MemListEntity through the updateRecord() function. Finally, MemListUIMan will update the list being displayed in MemGrid with function updateList().

Figure 5-3 sequence diagram illustrating use case Save Calculation to Memory List

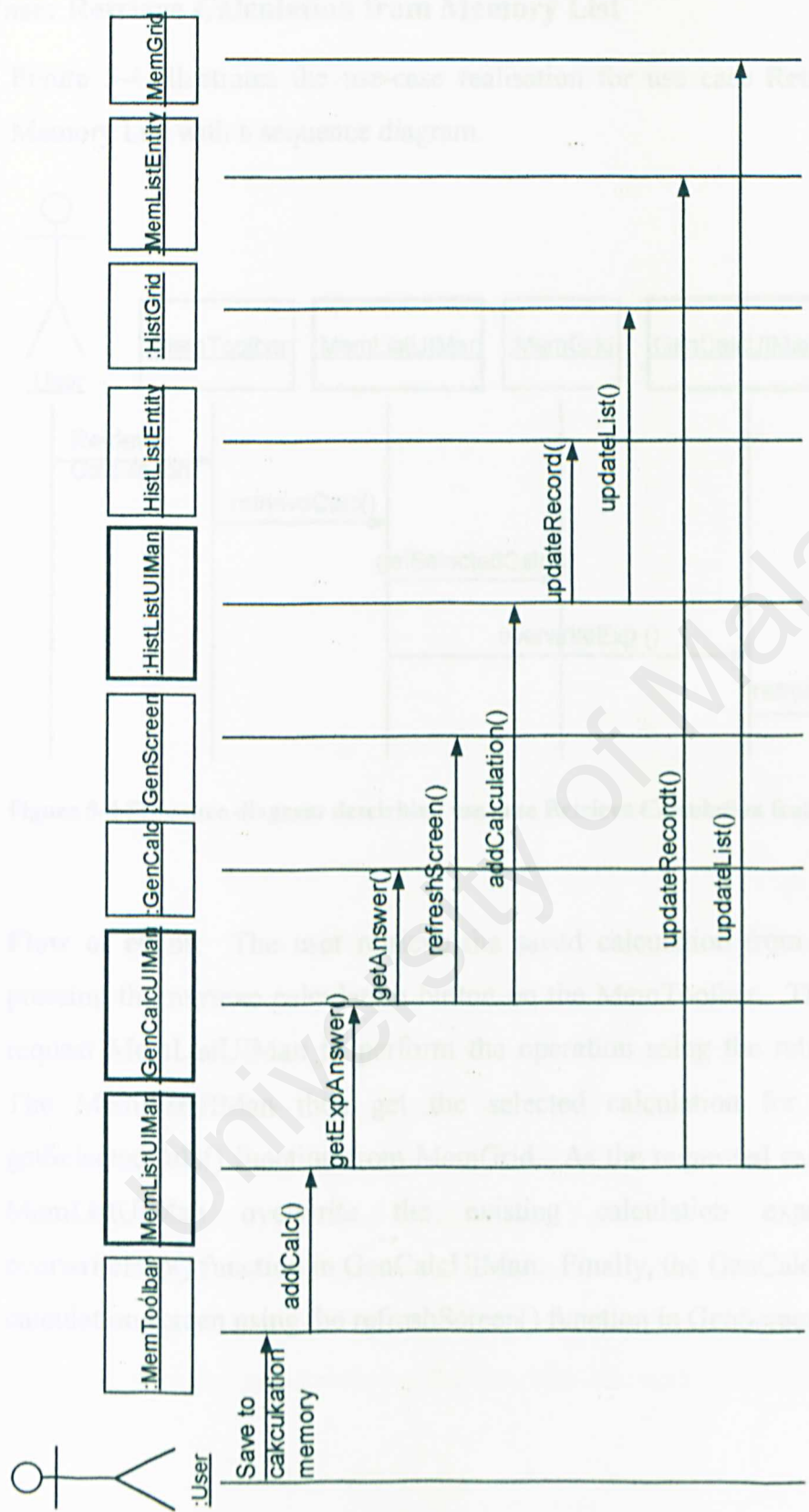


Figure 5-3 Sequence diagram describing use case Save Calculation to Memory List

Use Case: Retrieve Calculation from Memory List

Figure 5-4 illustrates the use-case realisation for use case Retrieve Calculation to Memory List with a sequence diagram.

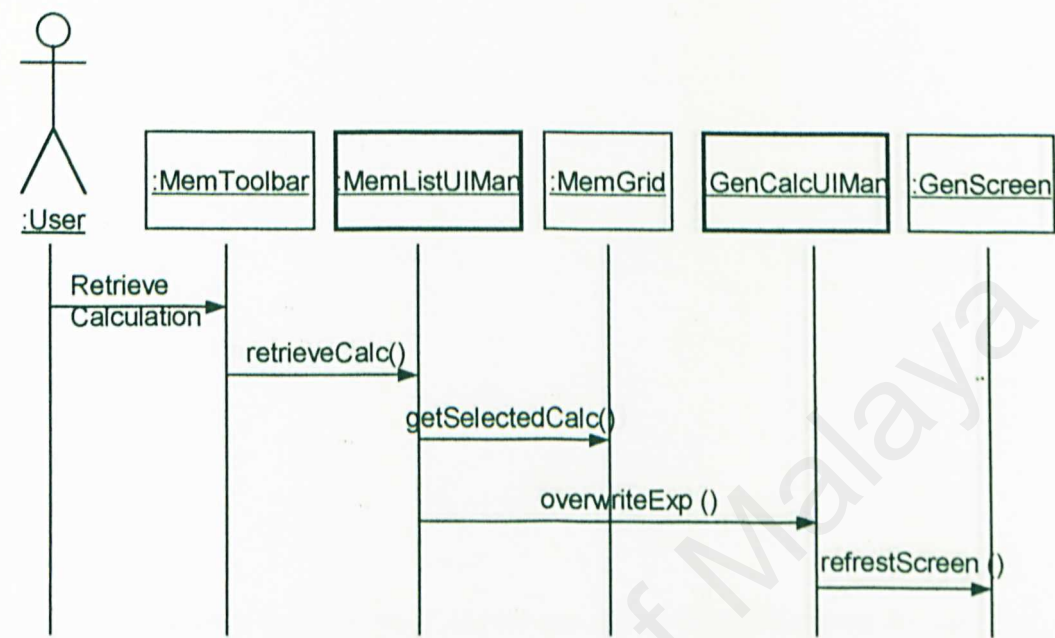


Figure 5-4 Sequence diagram describing use case Retrieve Calculation from Memory List

Flow of event: The user retrieve the saved calculation from the memory list by pressing the retrieve calculation button on the MemToolbar. The MemToolbar will request MemListUIMan to perform the operation using the retrieveCalc() function. The MemListUIMan then get the selected calculation for retrieval using the getSelectedCalc() function from MemGrid. As the requested calculation is received, MemListUIMan overwrite the existing calculation expression using the overwriteExp() function in GenCalcUIMan. Finally, the GenCalcUIMan refreshes the calculation screen using the refreshScreen() function in GenScreen.

Use Case: Retrieve Value from Memory List

Figure 5-5 illustrates the use-case realisation for use case Retrieve Value to Memory List with a sequence diagram.

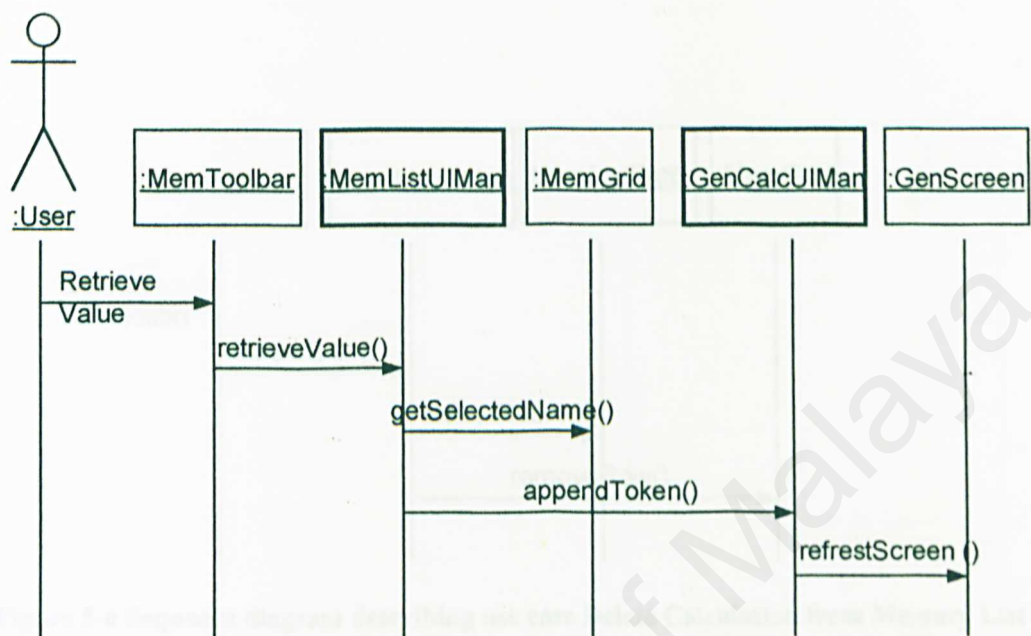


Figure 5-5 Sequence diagram describing use case Retrieve Value from Memory List

Flow of event: The user retrieve the value of a saved calculation from the memory list by pressing the retrieve value button on the MemToolbar. The MemToolbar will request MemListUIMan to perform the operation using the retrieveValue() function. The MemListUIMan then get the slot name for retrieval using the getSelectedSlotName() function from MemGrid. The slot name will be placed in the calculation expression instead of the value. As the requested calculation is received, MemListUIMan appends the retrieved slot name to the existing calculation expression using the appendToken() function in GenCalcUIMan. Finally, the GenCalcUIMan refreshes the calculation screen using the refreshScreen() function in GenScreen.

Use Case: Delete Calculation from Memory List

Figure 5-6 illustrates the use-case realisation for use case Delete Calculation to Memory List with a sequence diagram.

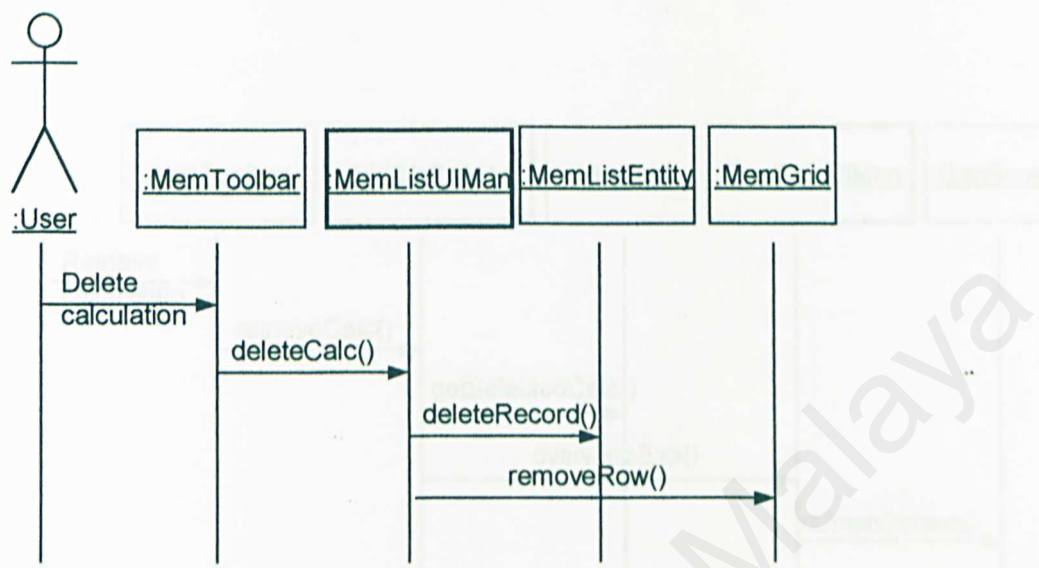


Figure 5-6 Sequence diagram describing use case Delete Calculation from Memory List

Flow of event: The user deletes a saved calculation from the memory list by pressing the delete calculation button on MemToolbar. The MemToolbar then request MemListUIMan to perform the operation using deleteCalc() function. Firstly, the MemListUIMan deletes the record in the database using deleteRecord() function with MemListEntity. Then, MemListUIMan remove the calculation from the list displayed in MemGrid using removeRow() function.

Use Case: Retrieve Calculation from History List

Figure 5-7 illustrates the use-case realisation for use case Retrieve Calculation to History List with a sequence diagram.

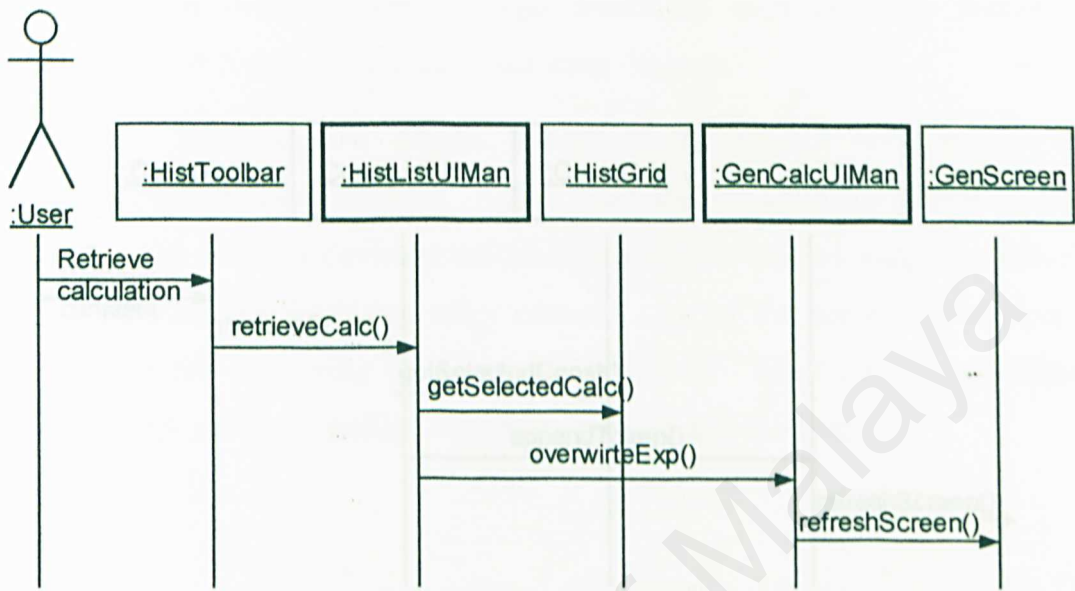


Figure 5-7 Sequence diagram describing use case Retrieve Calculation from History List

Flow of event: The user retrieve a calculation from the history list by pressing the retrieve calculation button on the HistToolbar. The HistToolbar will request HistListUIMan to perform the operation using the retrieveCalc() function. The MemListUIMan then get the selected calculation for retrieval using the getSelectedCalc() function from HistGrid. As the requested calculation is received, HistListUIMan overwrite the existing calculation expression using the overwriteExp() function in GenCalcUIMan. Finally, the GenCalcUIMan refreshes the calculation screen using the refreshScreen() function in GenScreen.

Use Case: Retrieve Constant from Constant List

Figure 5-8 illustrates the use-case realisation for use case Retrieve Constant to Constant List with a sequence diagram.

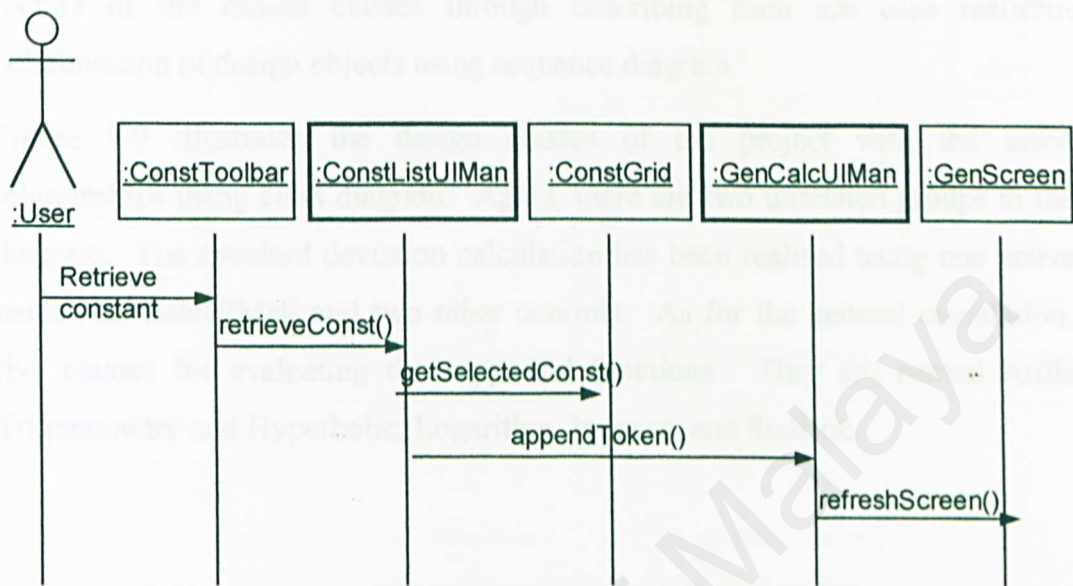


Figure 5-8 Sequence diagram describing use case Retrieve Constant from Constant List

Flow of event: The user retrieve a constant from the constant list by pressing the retrieve constant button on the ConstToolbar. The ConstToolbar will request ConstListUIMan to perform the operation using the `retrieveConst()` function. The ConstListUIMan then get the symbol for retrieval using the `getSelectedSymbol()` function from ConstGrid. The symbol will be placed in the calculation expression instead of the value. As the requested calculation is received, ConstListUIMan appends the retrieved symbol to the existing calculation expression using the `appendToken()` function in GenCalcUIMan. Finally, the GenCalcUIMan refreshes the calculation screen using the `refreshScreen()` function in GenScreen.

5.2 Design Class

This subsection specifies the design classes of this project. The results presented in this subsection are extracted from the previous subsection, which has provided a clear picture of the design classes through describing each use case realisation as collaboration of design objects using sequence diagram.

Figure 5-9 illustrates the design classes of the project with the association relationships using class diagram. Again, there are two unrelated groups in the class diagram. The standard deviation calculation has been realised using one active class named SDCalcUIMan and two other controls. As for the general calculation, there are five classes for evaluating the supported functions. They are named Arithmetic, Trigonometry and Hyperbolic, Logarithm, Indexes, and Statistic.



Figure 5-9 Class diagram showing the design classes

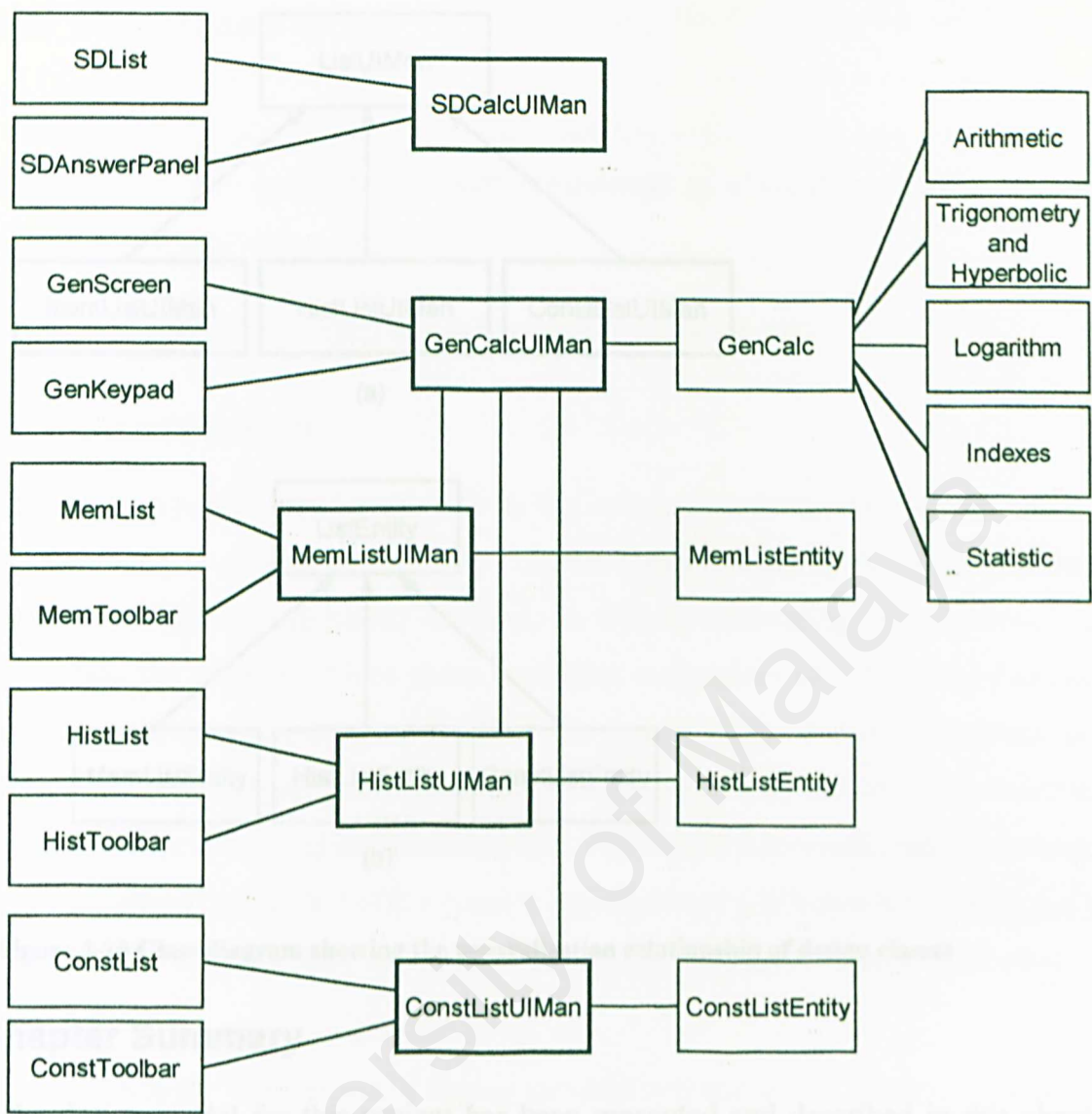
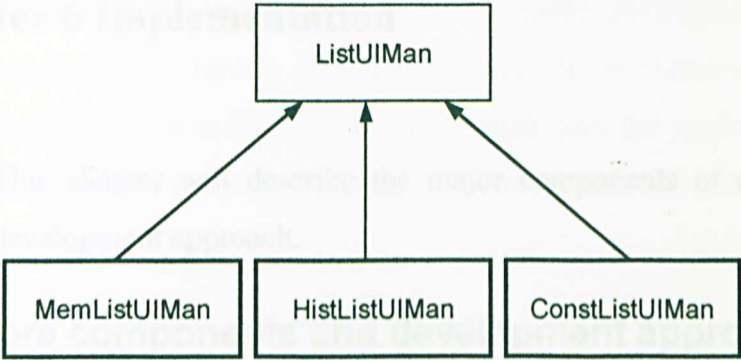
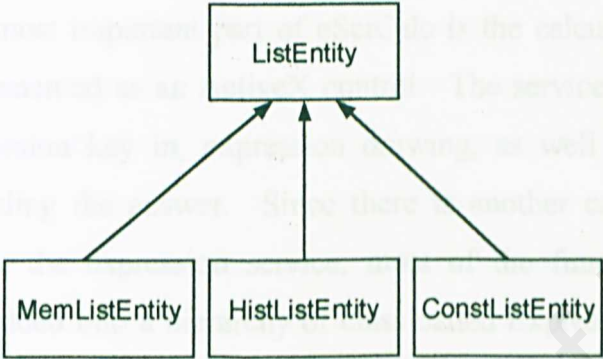


Figure 5-9 Class diagram showing the design classes



(a)



(b)

Figure 5-10 Class diagram showing the generalisation relationship of design classes

5.3 Chapter Summary

The design model for this project has been presented and described in this chapter. There are twenty-four design classes participated in the realisation of the eight use cases specified in the previous chapter.

5.4 References

[1] Jacobson, I., Booch, G., Rumbaugh, J.(1999).The Unified Software Development Process. United States: Addison-Wesley.

[2] Jacobson, I., Booch, G., Rumbaugh, J.(1999).The Unified Modelling Language User Guide. United States: Addison-Wesley.

Chapter 6 Implementation

This chapter will describe the major components of eSciCalc and their respective development approach.

6.1 Core components and development approach

6.1.1 Calculator screen

The most important part of eSciCalc is the calculator screen. This screen has been implemented as an ActiveX control. The services provided by this control includes expression key in, expression drawing, as well as calculating the expression and providing the answer. Since there is another component in the design that could utilise the expression service, most of the functionality of this control has been embedded into a hierarchy of class called Expression. This would enable more than one control to utilise this important service. Moreover, this would enable the services to be developed and tested with a console application or a Window application that the Visual C++ IDE provides debug services which the ActiveX control development does not.

In order to achieve the vision of having standard notation on the display, the font used would have to include numerous symbols that are used in mathematical calculations. Although all the symbols are included with the Unicode version of most Windows font, many of them are not included in the ASCII. Since one of the aim of this product is to run on Windows 9x and Windows 9x does not support Unicode, a custom font has been created. It is loaded when the calculator runs.

6.1.2 Expression classes

The expression class provides the fundamental form of any kind of expression. This class is an abstract class and is inherited by the GenCalc class in eSciCalc. The use of this hierarchy of classes for expression has also enable the support of complex number and base-n calculation easily.

The components of this class are mostly developed in a simple console mode application (wherever no graphical user interface is needed) or a simple MFC window application to make sure the way work and the working code has been developed before being integrated into the class being built since it is larger.

An expression object saves the expression it is handling as a string. String has been chosen as an expression might have to be stored into a database which does not support any user defined type directly. In addition, user defined type would not be possible to be passed in between an ActiveX control and its user.

This string representation of the expression will be transformed into a binary tree to perform expression drawing as well as expression calculation. A binary tree is used in this case as the expression could be easily tokenised according to its precedence. Moreover, the binary tree also automatically grouped the whole expression. For example, if a particular node is a fraction, we could easily identify its width, height, and so on since the left and right side of the node would be the fraction's numerator and denominator.. This would ease the expression drawing.

Besides that, the Expression class also store a list of variables and constant values as a linked list. This is where the values of variables could be assigned for calculation. This is implemented to support the use of memory list.

6.2 References

- [1] Deitel, H.M., Deitel, P.J.(1994).C: How to Program. United States: Prentice Hall.
- [2] Kruse, R.I., Tondo, C.L., Leung, B.P.(1997).Data Structure & Program Design in C. United States: Prentice Hall.
- [3] Sebesta, R.W.(1999).Concepts of Programming Languages,(4th ed). United States: Addison-Wesley.

Chapter 7 Testing

The purpose of testing is to

- Plan the tests required in each iteration, including integration tests and system tests. Integration tests are required for every build within the iteration, whereas system tests are required only at the end of the iteration.
- Design and implement the test by creating test cases that specify what to test, creating test procedures that specify how to perform the test, and creating executable test components to automate the test if possible.
- Perform the various tests and systematically handle the results of each test. Builds found to have defects are tested and possibly sent back to other core workflows, such as design and implementation, so that the significant defects can be fixed.[1]

This chapter present the result of testing for this project. There are four sections in this chapter. The first section, Section 7.1 will specify the input, result and conditions of use-case realisation of the design model. Section 7.2 describes the test procedure. The third section, Section 7.3 provides a summary of the chapter while, Section 7.4 states the references.

7.1 Test Case

A test case specify one way of testing the system, including what to test with which input or result, and under which conditions to test.

Use Case: Perform General Calculation

This test case will verify the key-in part of use case Perform General Calculation.

Input

- Key in the calculation statement $1 + (2 + 3) / \sin 20$ by pressing the keys on the GenKeypad panel.

Result

- The application should update the expression displayed.

Conditions

- No other use cases (instances) are allowed to access the General Calculation Interface during this test case.

This test case will verify the execute expression part of use case Perform General Calculation.

Input

- The equal key on the GenKeypad panel is pressed.

Result

- The General Calculation Interface should display 6.476780e as the answer.
- The history list should have the calculation.
- The database should update the record.

Condition

- No other use cases (instances) are allowed to access the General Calculation Interface during this test case.

Use Case: Perform Standard Deviation Calculation

This test case will verify the use case Perform Standard Deviation Calculation.

Input

- From the General Calculation Interface pressed the Standard Deviation Calculation button.
- Key in 3, 4, and 5 into the data list.

Result

- The number of data (n) should show a value of 3.
- The mean (\bar{x}) should show a value of 4.
- The population (σ_n) should show a value of 0.8165.
- The sample (σ_{n-1}) should show a value of 1.
- The Sum of value ($\sum x$) should show a value of 12.
- The Sum of Squares ($\sum x^2$) should show a value of 50.

Use Case: Save Calculation to Memory List

This test case will verify the use case Save Calculation to Memory List.

Input

- Key in the calculation statement $1 + (2+3) / \sin 20$ by pressing the keys on the GenKeypad panel.
- Press the memory list button on the toolbar above the screen.
- Pressed the save button on the Memory List Interface.

Result

- The memory list record will be update, which the Calculation column should show calculation statement $(1 + (2+3) / \sin 20)$ and the result of the calculation (6.476780e) should show in the result column.

Condition

- No redundancy is allowed.
- The memory list is not full.

Use Case: Retrieve Calculation from Memory List

This test case will verify the use case Retrieve Calculation from Memory List.

Input

- Press the retrieve calculation button on the MemToolbar.

Result

- The General Calculation Interface should show the selected calculation.
- If there is existing calculation expression on the General Calculation Interface, it should be overwriting.

Use Case: Retrieve Value from Memory List

This test case will verify the use case Retrieve Value from Memory List.

Input

- Press the retrieve value button on the MemToolbar.

Result

- The General Calculation Interface should show the selected calculation's value.
- If there is existing calculation expression or value, it should be overwriting.

Use Case: Delete Calculation from Memory List

This test case will verify the use case Delete Calculation from Memory List.

Input

- Press the delete calculation button on MemToolbar.

Result

- The record in the database should be deleting.
- The calculation displayed in the memory list should be deleting.

Use Case: Retrieve Calculation from History List

This test case will verify the use case Retrieve Calculation from History List.

Input

- Press the retrieve calculation button on the HistToolbar.

Result

- The General Calculation Interface should show the selected calculation.
- If there is existing calculation expression, it should be overwriting.

Use Case: Retrieve Constant from Constant List

This test case will verify the use case Retrieve Constant from Constant List.

Input

- Press the retrieve constant button on the ConsToolbar.

Result

- The General Calculation Interface should show the selected symbol.
- If there is existing calculation expression or value, it should be overwriting.

7.2 Test Procedure

A test procedure specifies how to perform one or several test cases or parts of them. A test procedure in this project will be an instruction for an individual on how to perform a test case manually.

Test case supported: calculation statement $1 + (2+3)/\sin 20$

1. From the General Calculation Interface, pressed the key on the GenKeypad panel follow the calculation statement $1 + (2+3)/\sin 20$.
2. Verify the display in the General Calculation Interface, it should display $1 + (2+3)/\sin 20$.
3. Press the equal key on the GenKeypad panel.
4. Verify the display in the General Calculation Interface; it should display 6.476780e as the answer.
5. Press the history list button on the toolbar above the screen.
6. Verify the following fields:
 - Calculation is $1 + (2+3)/\sin 20$.
 - Result is 6.476780e.

Test case supported: standard deviation calculation statement 3, 4, 5

1. From the General Calculation Interface pressed the Standard Deviation Calculation button.
2. Key in 3, 4, 5 into the data list.
3. Verify the following fields:
 - Number of data (n) is 3.
 - Mean (\bar{x}) is 4.
 - Variance (σ^2) is 0.66666667.
 - Standard deviation (σ) is 0.81649658.
 - Sum of value ($\sum x$) is 12.
 - Sum of Squares ($\sum x^2$) is 50.

7.3 Chapter Summary

The test model for this project has been presented and described in this chapter. There are nine test cases and two test procedures in this chapter. The test cases specify what to test in the system and the test procedures specify how to perform the test cases.

7.4 References

[1] Jacobson, I., Booch, G., Rumbaugh, J. (1999). The Unified Software Development Process. United States: Addison-Wesley.

Chapter 8 Evaluation

System evaluation is the process of identifying the strengths and weaknesses of the system and suggestions for possible future enhancements. This chapter present the project evaluation for this project. There are four sections in this chapter. The first section, Section 8.1 will specify the strength of the project. Section 8.2 describes the limitation of the project. The third section, Section 8.3 states the project's enhancements while Section 8.4 provides a summary of the chapter.

8.1 Strength

Among the strength of eSciCalc is:

- Runs on all 32-bit Windows Platform – Since Windows is currently the most popular and widely used operating system eSciCalc is designed to run on Windows platform.
- User-friendly Interface – eSciCalc provide a better user interface that would promote the use of standard notation, easy to redo, and reduce error rate and easy rechecking. Moreover, eSciCalc has a common Windows interface that the users have been used to.
- Ease of Control and Manipulation – eSciCalc can be used with the keyboard only, mouse only as well as a combination of both. This would provides all kinds of users with the method they prefer.
- Easy to Expand Architecture - eSciCalc design with an easy to expand architecture. This would enable the product to support a more complete set of calculations.

8.2 Limitation

There are limitations and weakness that could be improved in the future. Among the limitations are:

- Insufficient functionality – eSciCalc does not support adequate functions to be considered useful.

- Calculation value not large enough – although the range of the calculation value supported by eSciCalc is considerably large, it would be good if it could support a near infinite range to support the fast growing need of scientific calculation.
- The use of self-referenced structure instead of self-referenced class for the binary tree has made the code rather messy and difficult to expand. The future version could be overhauled to use self-referenced class.

8.3 Future Enhancements

Referring to eSciCalc's limitation given above, there is still high potential for future enhancements. These include:

- The most needed improvement would be the use of self-referenced class for the binary tree. The current method has been limiting the application's expansion capability. Moreover, it encapsulates the code better and less messy.
- Secondly, create or find a data type that would support higher value and precision like the one used in Microsoft® Calculator version 5.1 which seems infinite.
- Having a good framework already, more functions would be added to the general calculation mode of eSciCalc. Besides that, more calculation modes such as the complex number calculation would be added.

8.4 Chapter Summary

The evaluation and maintenance for this project has been presented and described in this chapter. The strength includes running on Windows platform, having user-friendly interface, ease of control and manipulation, and having an easy to expand architecture. As for the limitations, eSciCalc has limited functions, having a large but could be larger calculation value, and the construction of binary tree with structure instead of class.

Chapter 9 Conclusion

This chapter present the conclusion for this project. There are two sections in this chapter. The first section, Section 9.1 will specify the problems and solutions of this project. Section 9.2 describes knowledge gained and Section 9.3 state the conclusion.

9.1 Problems and Solutions

Various problems were occurred during the period in development eSciCalc. The previously mentioned problems and the solutions are undertaken to resolve them are described in the following sections.

- **Difficulty in Choosing a Suitable System Development Tools** - There are too many system development tools available to develop a stand-alone system currently. Therefore, choosing tools is very important in developing a system. The suitable tools can speed up System Development Life Cycle and to minimal the unexpected bug and error. To determine which suitable to use seeking advises and view from project supervisor and course mates engaging in similar project are carried out. Besides, surfing through the Internet and visiting the library help clarify some doubts.
- **How wide of the System Scope to be Build** - This is impossible to build a full-scale system because of time constraints. Also, inexperience and strange of the new programming language are contributed to this program (time consume in learning new programming language). Therefore, many discussions were help with the project supervisor and user to outline the scope of the project to be build. After the project scope has been defined, analysis of the system was done and the project started to develop.

9.2 Knowledge Gained

- Learned additional programming language, such as Visual C++.
- Learned how to model and develop software using Unified Modelling Language.
- Enriched experience in problem solving, especially on debugging errors.

- Improved on skills in writing documentation and reports.
- Learnt to work independently.
- Learnt the consequence of bad time management.

9.3 Conclusion

eSciCalc has achieved little objectives defined during the analysis phase but has provided the functions to be named a scientific calculator. In this project, eSciCalc focuses mainly on its user interface and calculation capabilities.

Building a scientific calculator is not at all easy although it seems easy. A lot of research, time and effort have been involved in making this project successful and in fulfilling the task's requirement. There was a lot of nudge and experience gained thought the development process of eSciCalc. One of the most essential knowledge gained from this project is the techniques on problem solving and knowledge on software development besides putting most of the programming method learnt these few years in to action. In additional, by developing this program, I have gained invaluable knowledge and experience. Besides that, I have enriched my knowledge and gained a lot of experience in system analysis, planning, design, implementation and testing. The development of this project by using the Software Engineering techniques will ease the tasks of future enhancements and experience.

Appendix A User Manual

Table of Content

Overview of eSciCalc..... 1

Features..... 1

Keyboard equivalents of calculator keys..... 4

General calculation..... 4

 Calculation..... 5

 Answer display format..... 9

Standard deviation calculation..... 9

Conclusion..... 10

A.1 Overview of eSciCalc

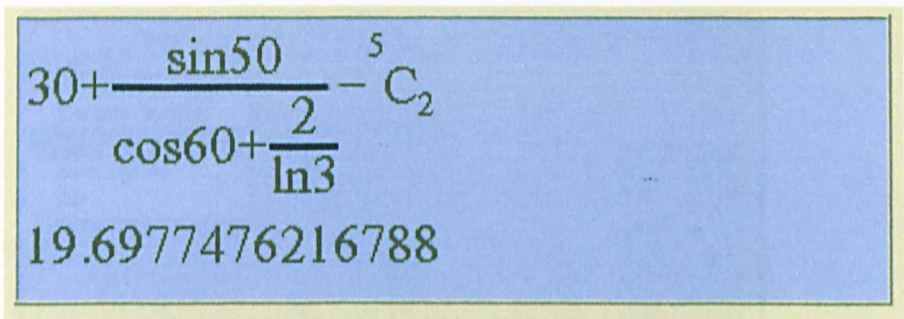
eSciCalc is a software-based scientific calculator. As the name suggest, it performs basic arithmetic calculations, such as addition, subtraction and multiplication, as well as scientific calculations, such as logarithm, trigonometric and hyperbolic.

This is the first version and does not have a very wide array of functions. However, all the basic scientific functions have been included.

A.2 Features

The features that eSciCalc provides are as follows:-

1. Calculation expression that user keys in is using mathematical standard notation. This makes the calculation to be performed clearly visible to the user. This eliminates key-in errors that are common in conventional calculator. Figure A-1 shows the standard notation display on the calculator screen.
2. eSciCalc has a dual line display to show the calculation expression and its answer at once. This enables the user to check the calculation expression with the answer they are provided. Figure A-1 shows eSciCalc’s screen with dual line display



The screenshot shows a blue calculator interface. The top display area contains the mathematical expression:
$$30 + \frac{\sin 50}{\cos 60 + \frac{2}{\ln 3}} - {}^5C_2$$
 Below the expression, the result is displayed as: 19.6977476216788

Figure A-1 eSciCalc calculator screen with standard notation and dual line display

3. All the available functions in eSciCalc could be keyed in using the keyboard or the mouse. Some might prefer the keyboard while some might prefer the mouse. Generally, keyboard input is faster than the mouse. However, switching between the mouse and keyboard is daunting too. Hence, eSciCalc provide full support for both.
4. The keypad of eSciCalc could be hidden and calculation being performed using the keyboard only. This feature is useful when the user is using eSciCalc with other application as hiding the keypad would provide a bigger visible area of the bottom application that the user wants to see. Figure A-2 shows keypad hiding.

Figure A-2 Action View Hide the keypad from the mouse to hide the keypad

5. eSciCalc could be not always on top. This feature helps the user to be able to work together with other application as well. It will enable the user to read the value provided by the calculator while working with other application.
6. eSciCalc provides a separate window to perform standard deviation. It enable the user to view the whole list of values together with their respective mean, variance, and so on. The list could also be manipulated easily.
7. eSciCalc provides answer formatting as well. The user can choose to have the answer shown in dot decimal form or the scientific form with the selected number of significant digit.

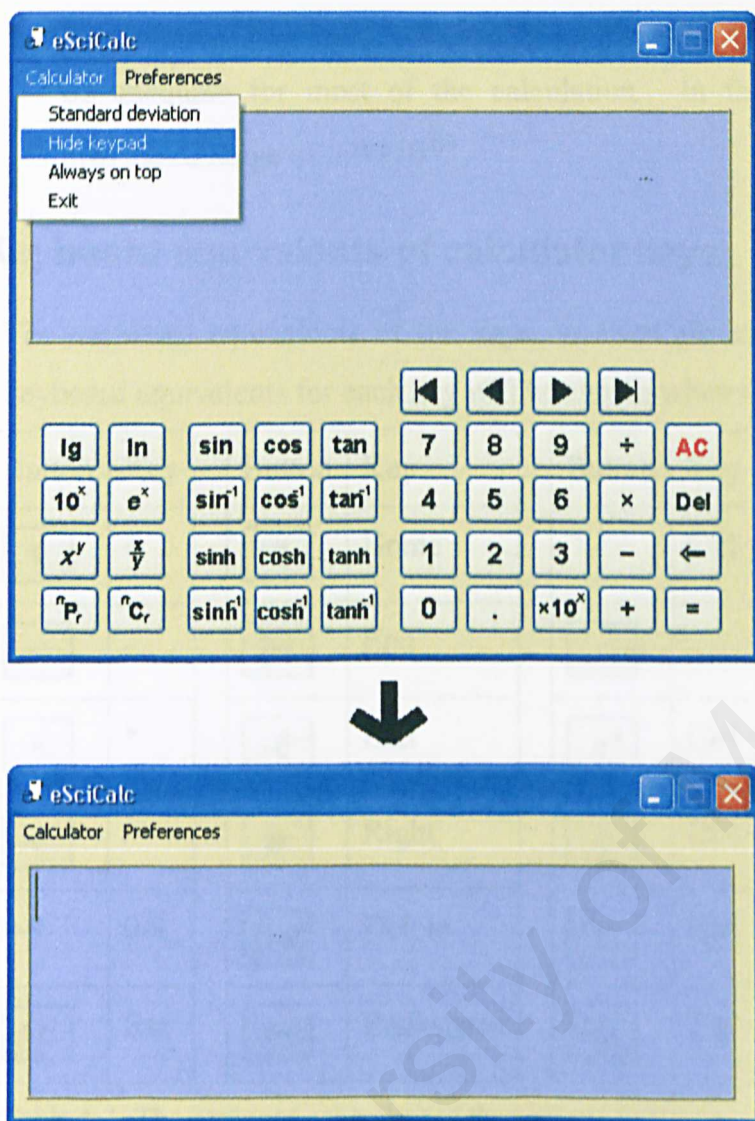


Figure A-2 Select Hide keypad from the menu to hide the keypad

5. eSciCalc can also be set always on top. This feature facilitate the calculate to be used together with other application as well. It will enable the user to read the value provided by the calculator while working with other application.
6. eSciCalc provides a separate window to perform standard deviation. It enable the user to view the whole list of values together with their respective mean, variance, and so on. The list could also be manipulated easily.
7. eSciCalc provides answer formatting as well. The user can chose to have the answer shown in the normal from or the scientific form with the selected number of significance digit.

8. The value that eSciCalc calculates has a range of $\pm 1.7 \times 10^{\pm 308}$. This range should be adequate for most of the calculation. In fact, most scientific calculator provides a range of $\pm 9 \times 10^{\pm 99}$.

A.3 Keyboard equivalents of calculator keys

The keyboard equivalents of the keys on eSciCalc are shown in Table A-1. The keyboard equivalents for each key will be shown when the mouse is pointing at it.


















Button	Key	Button	Key	Button	Key
	+		Home		Ctrl + A
	-		End		.
	*		Left		Ctrl + Shift + ^
	/		Right		Ctrl + F
0-9	0-9		Delete		Ctrl + P
	Esc		Backspace		Ctrl + C

Table A-1 The keyboard equivalent of the keys of eSciCalc.

A.4 General calculation

Figure A-3 shows the window for general calculation. This is where the expression-based calculations are performed. Expression-based calculation includes calculations that could be written or keyed in as an expression.

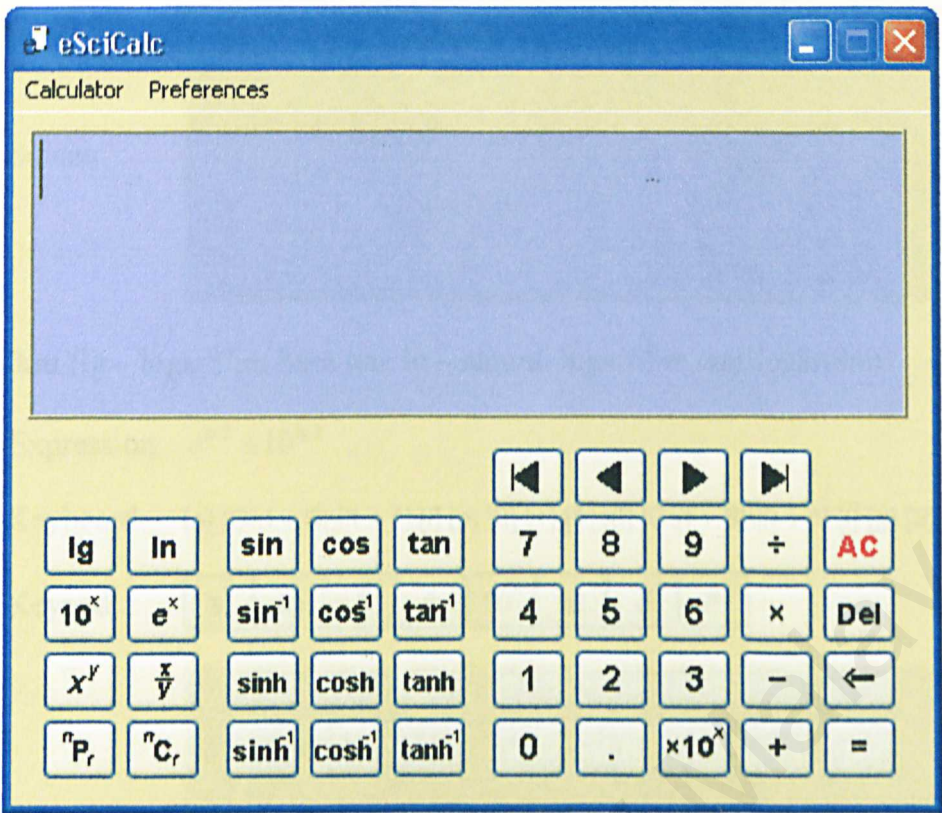


Figure A-3 General calculation window of eSciCalc.

The expression for this calculation mode is generally divided into linear and non-linear type. Linear expressions are calculations that could be written in a single line. This includes arithmetic, logarithm, trigonometric and hyperbolic. Non-linear expressions are expressions that could not be written in a single line if standard notation is employed. This category includes fraction, power, combination and permutation.

A.4.1 Calculation

Arithmetic (+, -, ×, ÷)

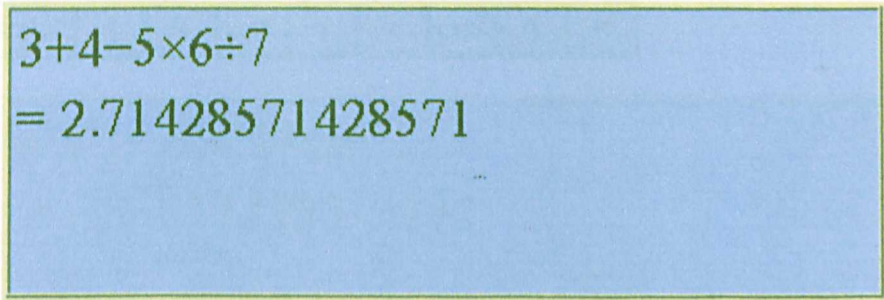
Expression: $3 + 4 - 5 \times 6 \div 7$

Keyboard: [3] [+] [4] [-] [5] [*] [6] [/] [7] [Enter]

Keypad:

3	+	4	-	5	×	6	÷	7	=
---	---	---	---	---	---	---	---	---	---

Screen:



$3+4-5\times 6\div 7$
 $= 2.71428571428571$

Logarithm (lg – logarithm base ten, ln – natural logarithm, antilogarithm)

Expression: $e^{\ln 3} + 10^{\lg 3}$

Keyboard: [e] [Ctrl + Shift + ^] [l] [n] [3] [+] [1] [0] [Ctrl + Shift + ^] [l] [g] [3] [Enter]

Keypad: 

Screen:



$e^{\ln 3} + 10^{\lg 3}$
 $= 6$

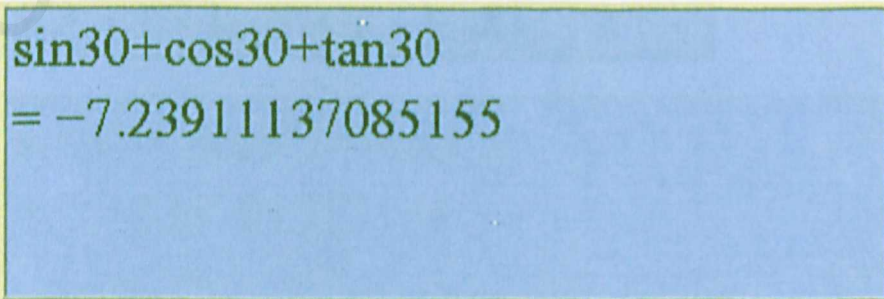
Trigonometric (sine, cosine, tangent, and their arcs)

Expression: $\sin 30 + \cos 30 + \tan 30$

Keyboard: [s] [i] [n] [3] [0] [+] [c] [o] [s] [3] [0] [+] [t] [a] [n] [3] [0] [Enter]

Keypad: 

Screen:



$\sin 30 + \cos 30 + \tan 30$
 $= -7.23911137085155$

Expression: $\sin^{-1} 1 + \cos^{-1} 1 + \tan^{-1} 1$

Keyboard: [s] [i] [n] [Ctrl + A] [1] [+] [c] [o] [s] [Ctrl + A] [1] [+] [t] [a] [n] [Ctrl + A] [1] [Enter]

Keypad: $\sin^{-1} 1 + \cos^{-1} 1 + \tan^{-1} 1 =$

Screen:

$$\sin^{-1} 1 + \cos^{-1} 1 + \tan^{-1} 1 = 2.35619449019234$$

Hyperbolic (hyperbolic sine, cosine, tangent, and their arcs)

Expression: $\sinh 3 + \cosh 3 + \tanh 3$

Keyboard: [s][i][n][h][3][+][c][o][s][h][3][+][t][a][n][h][3][Enter]

Keypad: $\sinh 3 + \cosh 3 + \tanh 3 =$

Screen:

$$\sinh 3 + \cosh 3 + \tanh 3 = 21.0805916768744$$

Expression: $\sinh^{-1} 1 + \cosh^{-1} 1 + \tanh^{-1} 0.5$

Keyboard: [s][i][n][h][Ctrl + A][1][+][c][o][s][h][Ctrl + A][1][+][t][a][n][h][Ctrl + A][.][5][Enter]

Keypad: $\sinh^{-1} 1 + \cosh^{-1} 1 + \tanh^{-1} . 5 =$

Screen:

$$\sinh^{-1} 1 + \cosh^{-1} 1 + \tanh^{-1} . 5 = 1.4306797313536$$

Fraction

Expression: $1 + \frac{5}{6}$
 $\frac{1 + \frac{5}{6}}{7}$

Keyboard: [Ctrl + F] [1] [+] [Ctrl + F] [5] [Right] [6] [Right] [7] [Enter]

Keypad: $\frac{x}{y}$ 1 + $\frac{x}{y}$ 5 \blacktriangleright 6 \blacktriangleright 7 =

Screen:

$$\frac{1 + \frac{5}{6}}{7} = 0.261904761904762$$

Power

Expression: $2^{2^2} + 8^{\frac{1}{3}}$

Keyboard: [2] [Ctrl + Shift + ^] [2] [Ctrl + Shift + ^] [2] [End] [+] [8] [Ctrl + Shift + ^] [Ctrl + F] [1] [Right] [3] [Enter]

Keypad: 2 x^y 2 x^y 2 \blacktriangleright + 8 x^y $\frac{x}{y}$ 1 \blacktriangleright 3 =

Screen:

$$2^{2^2} + 8^{\frac{1}{3}} = 18$$

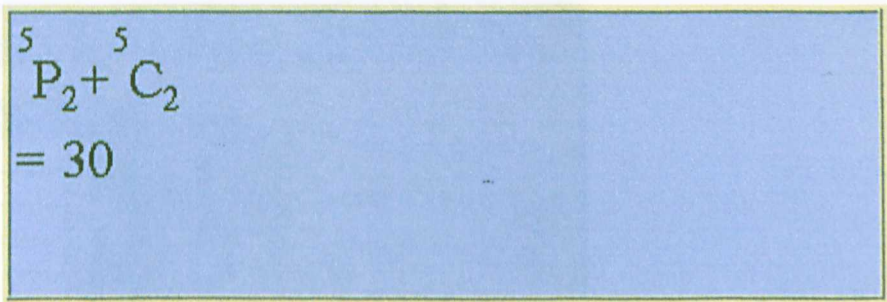
Combination and Permutation

Expression: ${}^5P_2 + {}^5C_2$

Keyboard: [Ctrl + P] [5] [Right] [2] + [Ctrl + C] [5] [Right] [2] [Enter]

Keypad: nP_r 5 \blacktriangleright 2 + nC_r 5 \blacktriangleright 2 =

Screen:



A screenshot of a calculator screen with a blue background. It displays the calculation of the sum of two combinations: ${}^5P_2 + {}^5C_2$. The result shown is 30.

A.4.2 Answer display format

The answer display filed could be set according to the user’s preference. There are two format provided by eSciCalc. The answer display format is specified using the menu under Preference → Answer style.

Normal

The answer would be display without specific format and depending on the figure. If the value is too large and the exponent part exists, it will be shown.

Scientific

The answer would be displayed according to the number of significance digits specified. If the answer does not have that many significance digits, it zero would be placed and if there are more digits than the specified significance digits, it would be rounded to the number of significance digits wanted.

A.5 Standard deviation calculation

eSciCalc provides standard deviation calculation support through a separate window (see Figure A-4). This separate window shows the whole list of values in the sample to be calculated. This has also let the user to be sure that the answer they get is the one they wanted. Another advantage is the ability to edit the list easily in case of error or similar list that they need to calculate.

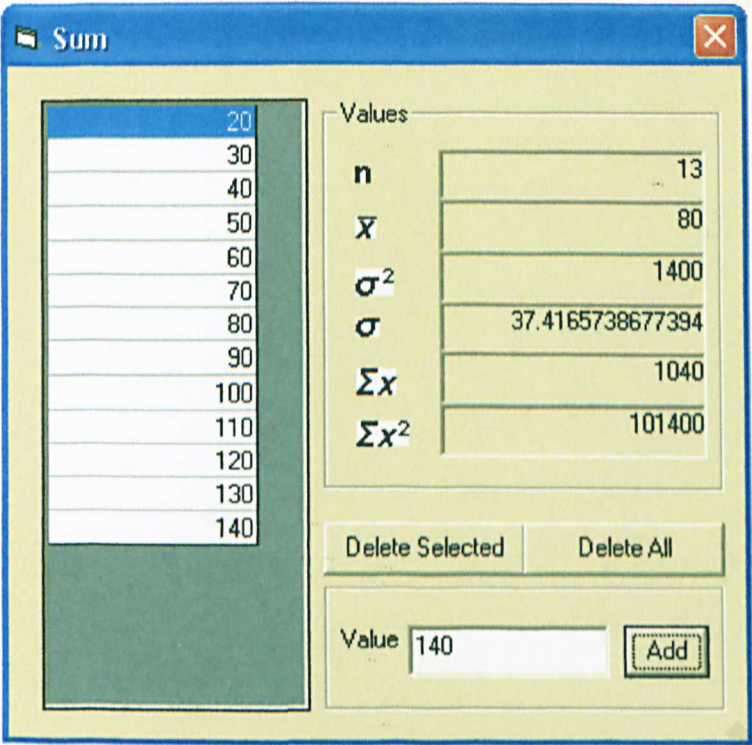


Figure A-4 Standard deviation calculation window of eSciCalc.

The bottom right corner of the window sited the controls that is used to manipulate the list. The button labelled “Delete All” is used to clear the list. As for deleting only a particular value, select the value with the mouse and press the button labelled “Delete Selected”

In order to add values to the list, fill the text box next to the label value with the number wanted to add and press the button to its right labelled “Add”.

Every time the content of the list changes, the six values will be recalculated automatically.

A.6 Conclusion

Thank you for using eSciCalc.